ПРЕДСТАВЛЕНИЕ ПРОСТРАНСТВЕННЫХ ФОРМ

Лекция 7

Иванова Юлия Александровна

Представление пространственных форм

Трёхмерная форма объектов

Ранее мы представляли пространственные объекты в ***виде последовательности отрезков прямых***, заданных в мировых координатах.

Но! С помощью отрезков невозможно определить даже криволинейные поверхности.

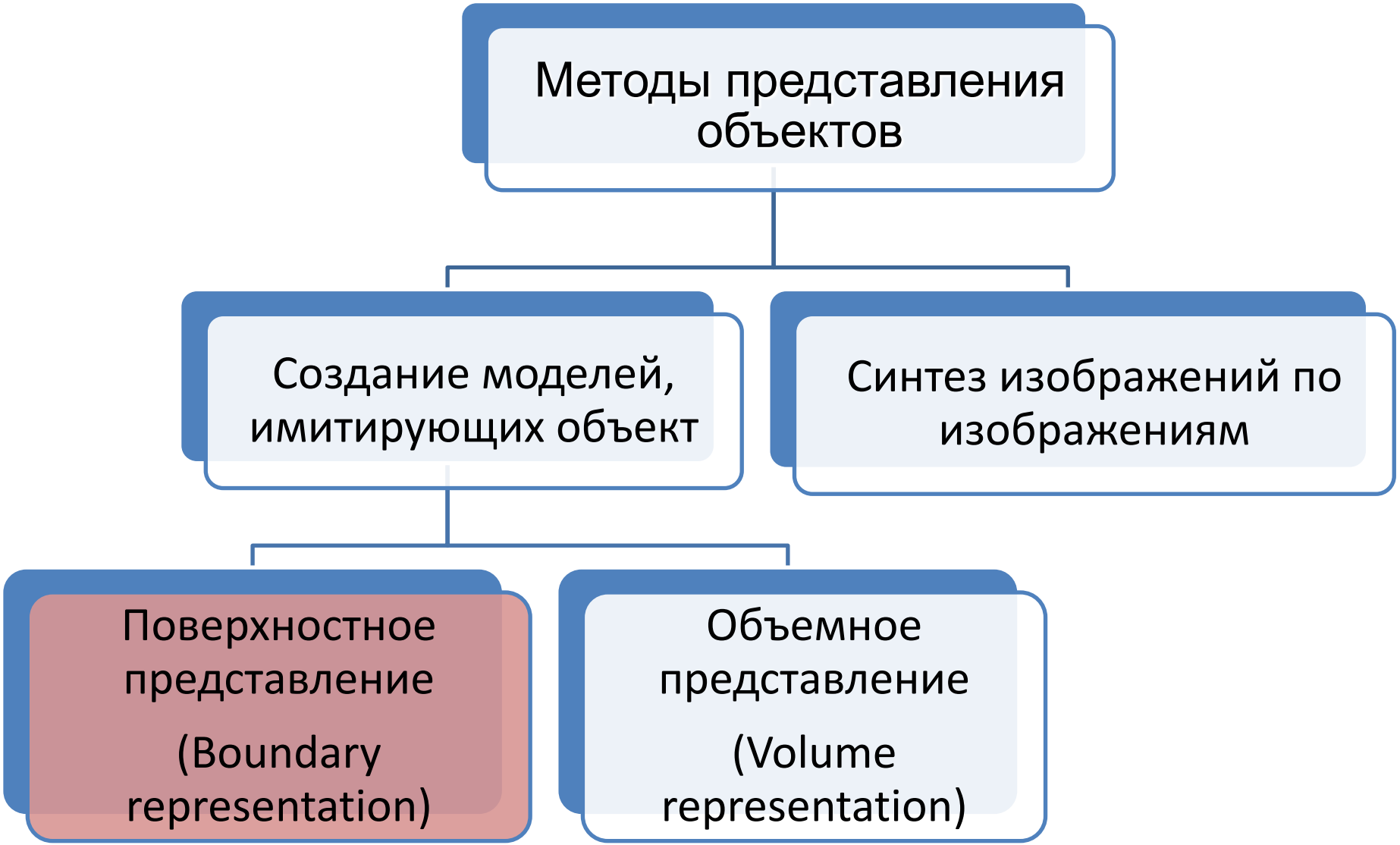
Представление пространственных форм

- Для построения 3D объекта необходимо иметь модели, включающие характеристики:
  - Положения тела в пространстве, его размер, форма и т.д.
  - Информацию о поверхности тела (цвет, текстура, отражательная способность, прозрачность и т.д.)
  - Информацию о материале объектов (плотность, масса, возможность деформироваться).

Основные типы представлений объектов в 3D пространстве

1. Создание модели, имитирующей 3D объект, ее отображение ее на экран:
   - поверхностное представление (хранится только граница объекта);
   - объемное представление (хранится информации обо всех точках объекта).
2. Синтез изображения трехмерных объектов по уже существующим изображениям:

Но! ***Метод синтеза изображений*** по изображениям не позволяет осуществлять никаких действий, кроме визуализации объектов!


Методы представления объектов
Создание моделей, имитирующих объект
Синтез изображений по изображениям
Поверхностное представление
(Boundary representation)
Объемное представление
(Volume representation)

Создание моделей, имитирующих объект

1. Поверхностное представление

- Объект создается при помощи набора тонких поверхностей, составляющих его границу.
- Нет необходимости обрабатывать внутренность тела:
  - ***дизайнерские проекты***,
  - ***моделирование обводов*** изделия,
  - создание ***объектов с нестандартными элементами*** (например, скругление с изменяемым радиусом).
  - и др.

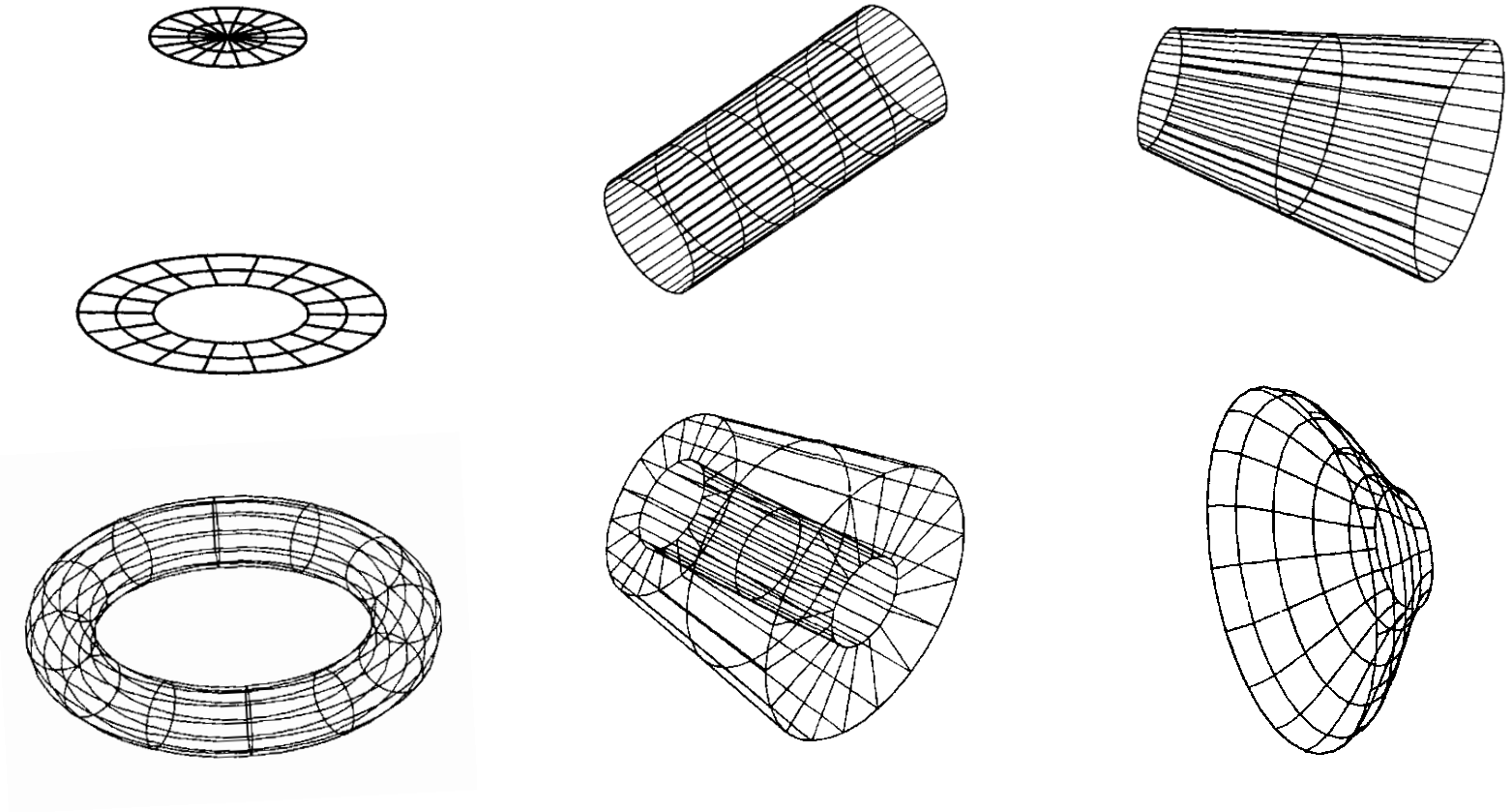
1. Поверхностное представление

Существует несколько основных способов задания границы тела.

- **Поверхность вращения**
- **Явная функция** $z = f(x, y)$
- **Неявная функция**. $f(x, y, z) = 0$
- **Параметрическое задание, сплайны, NURBS.** $\vec{P} = \vec{F}(u, v)$
- **Полигональные поверхности.**
- **Конструктивная геометрия тел** (Constructive Solid Geometry, CSG).

Поверхности вращения

- Вращение двумерного объекта вокруг некоторой оси в пространстве.



Поверхности вращения(1)

Параметрическое уравнение точки на поверхности вращения можно получить, если вспомнить, что параметрическое уравнение вращаемого объекта, например

$$P(t) = [x(t) \quad y(t) \quad z(t)] \quad 0 \le t \le t_{\max},$$

есть функция одного параметра t. Вращение вокруг оси приводит к тому, что координаты зависят также от угла поворота. Таким образом, точка на поверхности вращения определяется двумя параметрами t и ϕ. Как показано на рис. 6-5, это бипараметрическая функция.

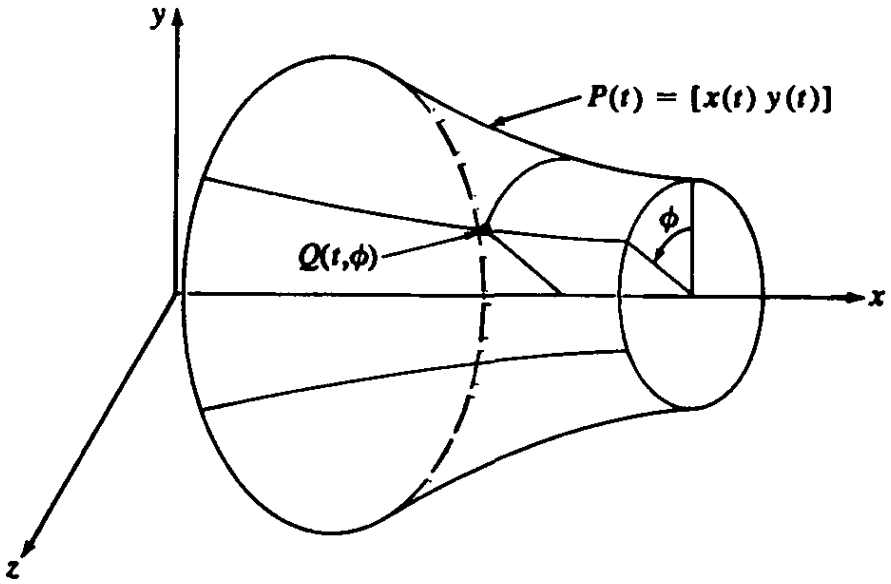


Рис. 6-5 Бипараметрическая поверхность вращения.

Поверхности вращения(2)

Заметим, что $Q(t, \phi)$ являетя вектор-функцией, т. е. в векторной форме

$$Q(t, \phi) = x(t)\mathbf{i} + y(t)\cos\phi\mathbf{j} + y(t)\sin\phi\mathbf{k},$$

где $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ — единичные векторы в x, y, z направлениях, соответственно.

Для рассматриваемого частного случая, т. е. вращения вокруг оси x объекта, расположенного в плоскости xy, уравнение поверхности записывается[1]

$$Q(t, \phi) = [x(t) \quad y(t)\cos\phi \quad y(t)\sin\phi] \tag{6-1}$$

Заметим, что здесь координата x не меняется. В качестве иллюстрации приведем пример.

Пример 6-1 Простая поверхность вращения

Рассмотрим отрезок с концами $P_1[1 \quad 1 \quad 0]$ и $P_2[6 \quad 2 \quad 0]$, лежащий в плоскости xy. Вращение отрезка вокруг оси x породит коническую поверхность. Определим на поверхности координаты точки с параметрами $t = 0.5$, $\phi = \pi/3$ $(60°)$.

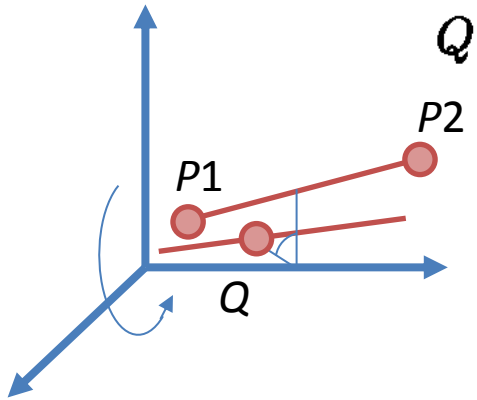
Параметрическое уравнение отрезка, соединяющего P_1 и P_2, имеет вид

$$P(t) = [x(t) \quad y(t) \quad z(t)] = P_1 + (P_2 - P_1)t \qquad 0 \le t \le 1$$

с декартовыми координатами

$$\begin{aligned} x(t) &= x_1 + (x_2 - x_1)t = 1 + 5t, \\ y(t) &= y_1 + (y_2 - y_1)t = 1 + t, \\ z(t) &= z_1 + (z_2 - z_1)t = 0. \end{aligned}$$

Используя уравнение (6-1), получим точку $Q(1/2, \pi/3)$ на поверхности вращения

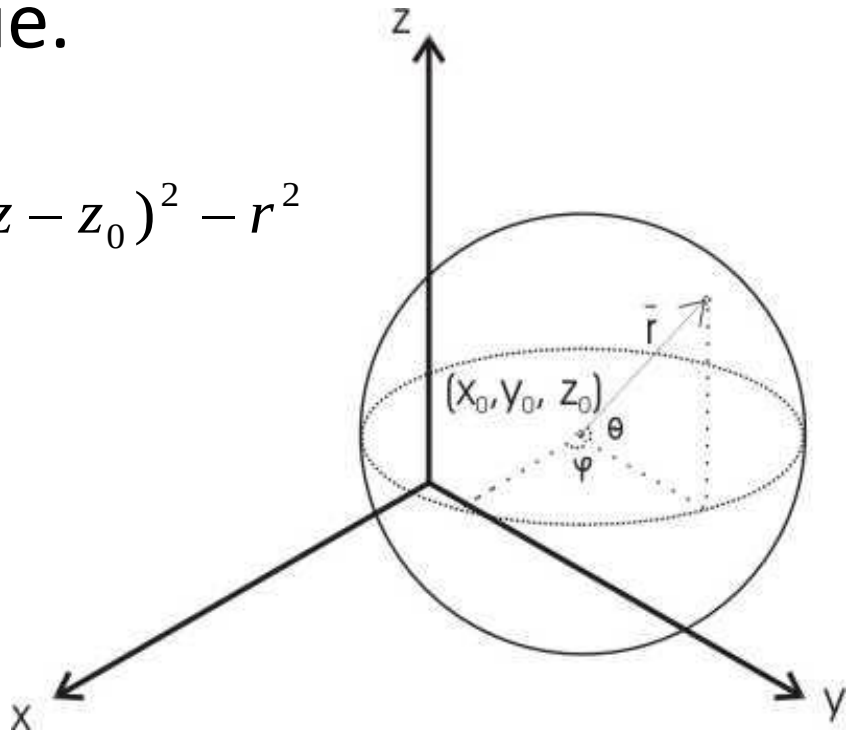


$$\begin{aligned} Q(1/2, \pi/3) &= [1 + 5t \quad (1 + t)\cos\phi \quad (1 + t)\sin\phi] \\ &= \left[\frac{7}{2} \quad \frac{3}{2}\cos\left(\frac{\pi}{3}\right) \quad \frac{3}{2}\sin\left(\frac{\pi}{3}\right)\right] \\ &= \left[\frac{7}{2} \quad \frac{3}{4} \quad \frac{3\sqrt{3}}{4}\right] = [3.5 \quad 0.75 \quad 1.3]. \end{aligned}$$

Представление объектов с помощью ***неявной функции***

- Указываем некую функцию, нулями которой будут точки, образующие поверхность, либо дающие ее приближение.

$$F(x, y, z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2$$



Сфера

Представление объектов с помощью параметрического задания

При параметрическом задании координаты точек поверхности рассматриваются как некие функции от двух параметров, пробегающих некоторый набор значений.

$$\begin{cases} x = x(u,v) \\ y = y(u,v) \\ z = z(u.v) \end{cases}$$

Так, например, параметризация сферы относительно двух углов φ (долгота) и θ (широта) будет иметь вид:

$$\begin{cases} x = r\cos\theta\cos\varphi \\ y = r\cos\theta\sin\varphi, \quad 0 \le \varphi < 2\pi, \; -\dfrac{\pi}{2} \le \theta \le \dfrac{\pi}{2} \\ z = r\sin\theta \end{cases}$$

Помимо этого используются различного рода особые параметрические поверхности, например, поверхности Безье, B-сплайны, в частности, NURBS (неоднородный рациональный B-сплайн).

Билинейные поверхности

Конструируется из 4-х угловых точек единичного квадрата в параметрическом пространстве P(0,0), P(0,1), P(1,1), P(1,0). Любая точка внутри параметрического квадрата задается формулой:

$$Q(u,w) = P(0,0)(1-u)(1-w) + P(0,1)(1-u)w + P(1,0)u(1-w) + P(1,1)uw.$$

В матричном виде

$$Q(u,w) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} P(0,0) & P(0,1) \\ P(1,0) & P(1,1) \end{bmatrix} \begin{bmatrix} 1-w \\ w \end{bmatrix}.$$

Базовые точки заданы в трехмерном пространстве -> билинейная поверхность будет трехмерна.

Базовые точки не лежат в одной плоскости -> билинейная поверхность не лежит ни в какой плоскости.

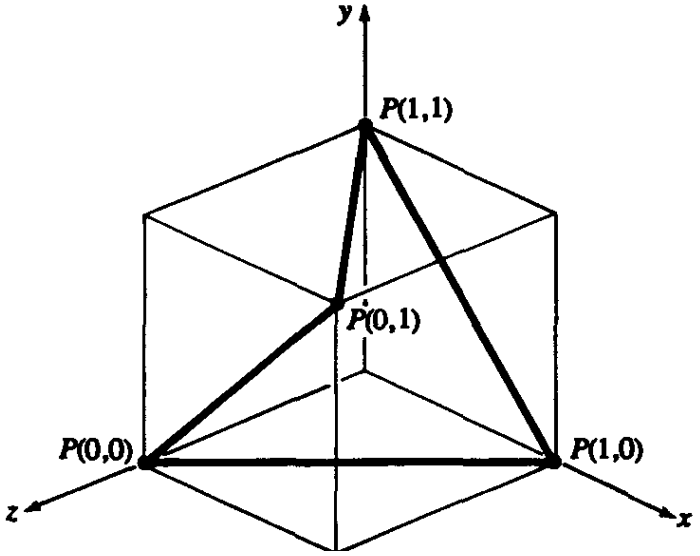
Пример билинейной поверхности

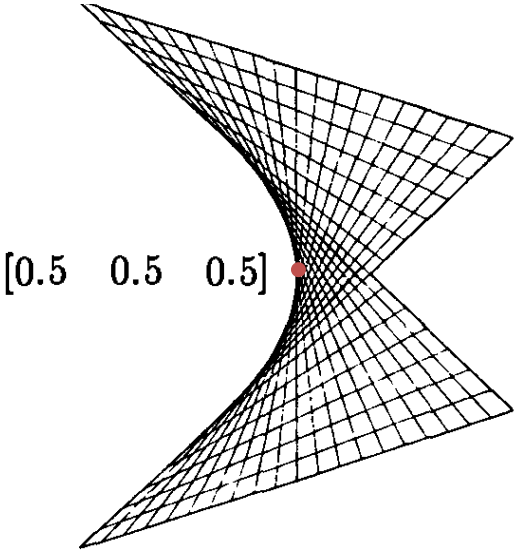
Найти точку на билинейной поверхности, заданной точками $P(0,0) = [0 \quad 0 \quad 1]$, $P(0,1) = [1 \quad 1 \quad 1]$, $P(1,0) = [1 \quad 0 \quad 0]$, $P(1,1) = [0 \quad 1 \quad 0]$, т. е. концами противоположных диагоналей, лежащих на противоположных гранях единичного куба в объектном пространстве. Искомая точка имеет координаты $u = w = 0.5$ в параметрическом пространстве.

Напомним, что поверхность в объектном пространстве является векторной функцией:

$$Q(u,w) = [x(u,w) \quad y(u,w) \quad z(u,w)]$$

$$\begin{aligned} Q(0.5,0.5) &= [0 \quad 0 \quad 1](1-0.5)(1-0.5) + [1 \quad 1 \quad 1](1-0.5)(0.5) \\ &+ [1 \quad 0 \quad 0](0.5)(1-0.5) + [0 \quad 1 \quad 0](0.5)(0.5) \\ &= 0.25[0 \quad 0 \quad 1] + 0.25[1 \quad 1 \quad 1] \\ &+ 0.25[1 \quad 0 \quad 0] + 0.25[0 \quad 1 \quad 0] \\ &= [0.5 \quad 0.5 \quad 0.5] \end{aligned}$$



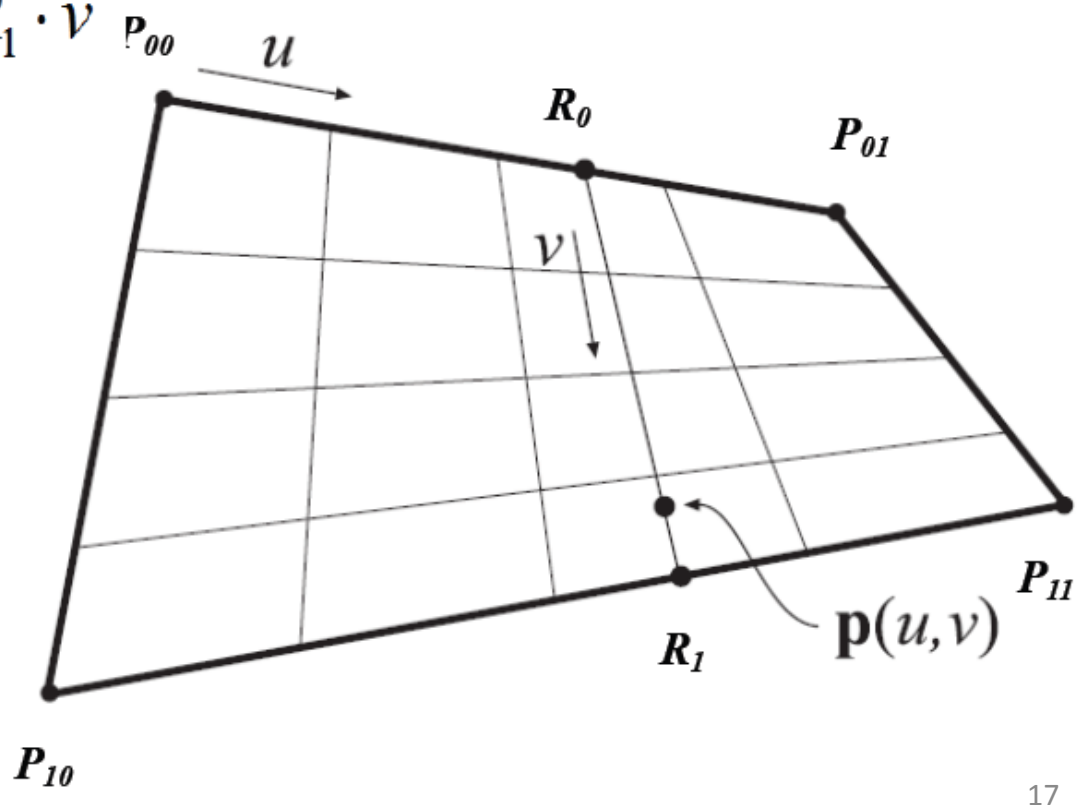


Поверхности Безье: билинейные

$$R_0 = P_{00} \cdot (1-u) + P_{01} \cdot u$$

$$R_1 = P_{10} \cdot (1-u) + P_{11} \cdot u$$

$$P(u,v) = R_0 \cdot (1-v) + R_1 \cdot v$$

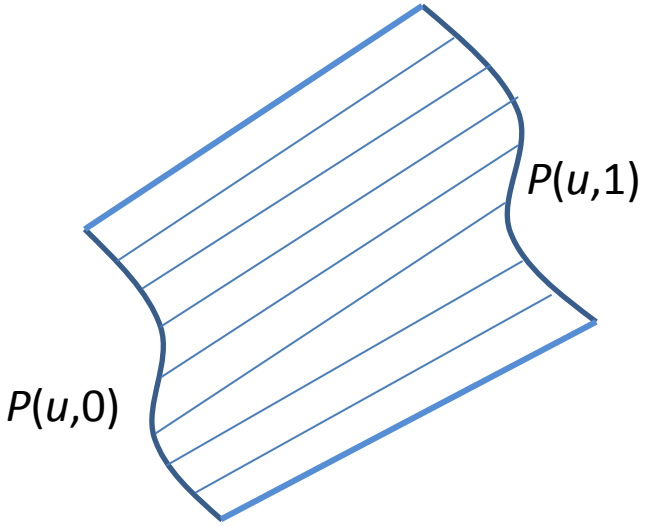


«!» Каждая изопараметрическая линия на билинейной поверхности является прямой линией.

Линейчатые и развертывающиеся поверхности

Задаются уравнением: $Q(u,w) = P(u,0)(1-w) + P(u,1)w$

или $$[Q] = [x(x,w) \quad y(u,w) \quad z(u,w)] = [1-w \quad w] \begin{bmatrix} P(u,0) \\ P(u,1) \end{bmatrix}$$



Поверхности Кунса

Пусть известны четыре граничные кривые $P(u,0)$, $P(u,1)$, $P(0,w)$, $P(1,w)$ билинейная смешивающая функция.

$$Q(u,w) = P(u,0)(1-w) + P(u,1)w + P(0,w)(1-u) + P(1,w)u.$$

Однако, проверив этот результат в угловых точках куска поверхности, например

$$Q(0,0) = P(0,0) + P(0,0) = 2P(0,0)$$

и на границах, например

$$Q(0,w) = P(0,0)(1-w) + P(0,1)w + P(0,w)$$

т.к. угловые точки учитываются дважды

Правильный результат можно получить с помощью вычитания дополнительных членов, возникающих из-за удвоения угловых точек:

$$\begin{aligned} Q(u,w) = & P(u,0)(1-w) + P(u,1)w + P(0,w)(1-u) + P(1,w)u \\ & - P(0,0)(1-u)(1-w) - P(0,1)(1-u)w \cdot \\ & - P(1,0)u(1-w) - P(1,1)uw. \end{aligned} \qquad (6\text{-}49)$$

Линейная поверхность Кунса

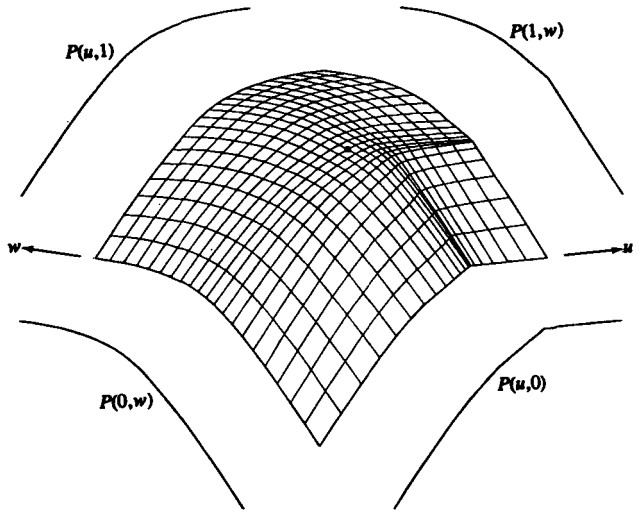
В матричной форме уравнение (6-49) имеет вид:

$$Q(u,w) = [1-u \quad w] \begin{bmatrix} P(0,w) \\ P(1,w) \end{bmatrix} + [P(u,0) \quad P(u,1)] \begin{bmatrix} 1-w \\ w \end{bmatrix}$$

$$- [1-u \quad w] \begin{bmatrix} P(0,0) & P(0,1) \\ P(1,0) & P(1,1) \end{bmatrix} \begin{bmatrix} 1-w \\ w \end{bmatrix}$$

или более компактно

$$Q(u,w) = [1-u \quad u \quad 1] \begin{bmatrix} -P(0,0) & -P(0,1) & P(0,w) \\ -P(1,0) & -P(1,1) & P(1,w) \\ P(u,0) & P(u,1) & 0 \end{bmatrix} \begin{bmatrix} 1-w \\ w \\ 1 \end{bmatrix}.$$



Бикубическая поверхность Кунса

Для всех четырех граничных кривых куска бикубической поверхности Кунса используются нормализованные кубические сплайны Для задания внутренней части куска используются кубические смешивающие функции. Таким образом, каждую граничную кривую можно представить в общем виде

$$P(t) = B_1 + B_2 t + B_3 t^2 + B_4 t^3, \quad 0 \le t \le 1. \tag{5-2}$$

Для одного сегмента нормализованного кубического сплайна с известными касательными и координатными векторами на концах, каждая из четырех граничных кривых, $P(u,0)$, $P(u,1)$, $P(0,w)$ и $P(1,w)$ задается следующим образом .

$$P(t) = [T][N][G] =$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_1' \\ P_2' \end{bmatrix} \quad 0 \le t \le 1, \tag{5-27}$$

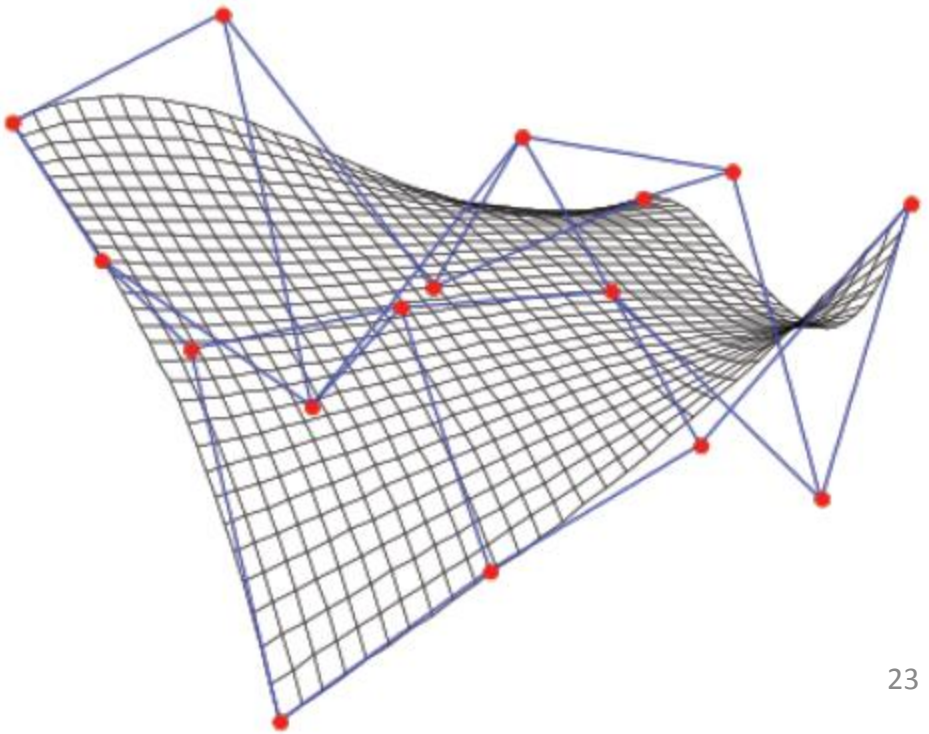
где t становится соответственно u или w, а P_1, P_2, P_1', P_2' — координатные и касательные векторы на концах соответствующей граничной кривой

Поверхности Безье: общий случай

$$B(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij} \cdot \mathbf{b}_{j,m}(u) \cdot \mathbf{b}_{i,n}(v), \quad u,v \in [0,1]$$

$$\mathbf{b}_{i,n}(t) = C_n^i \cdot t^i \cdot (1-t)^{n-i}$$

$$C_n^i = \binom{n}{i} = \frac{n!}{i!(n-i)!}$$

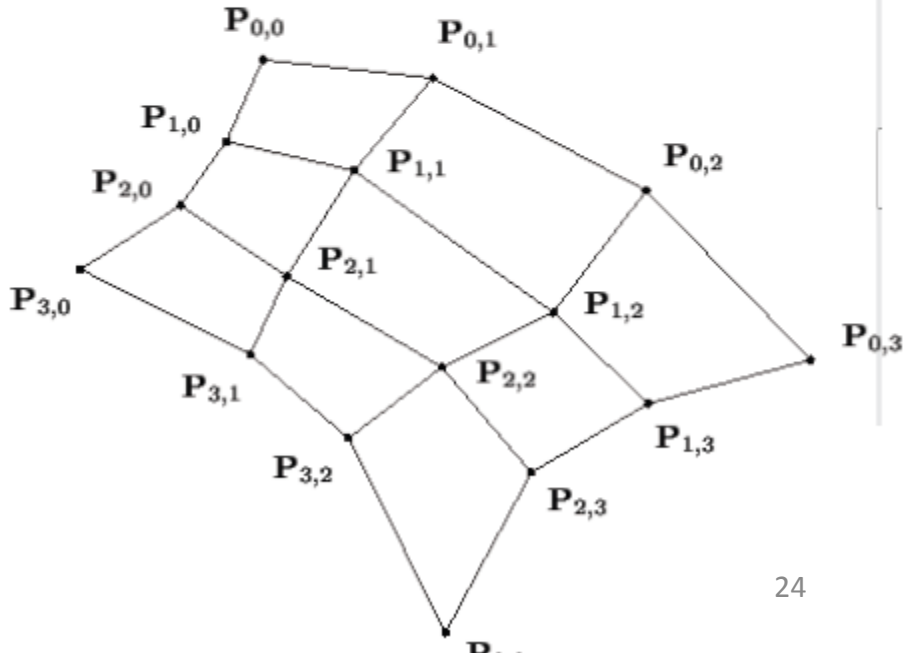


Каждая из граничных кривых поверхности Безье является кривой Безье.

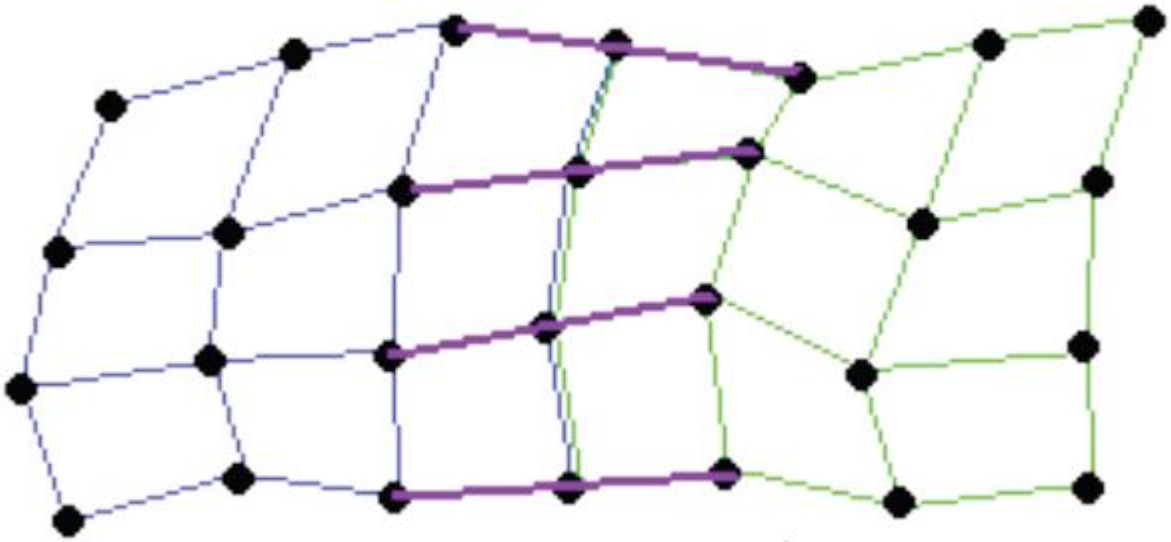
Бикубическая поверхность Безье

$$B(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_B \cdot \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \cdot M_B^T \cdot \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

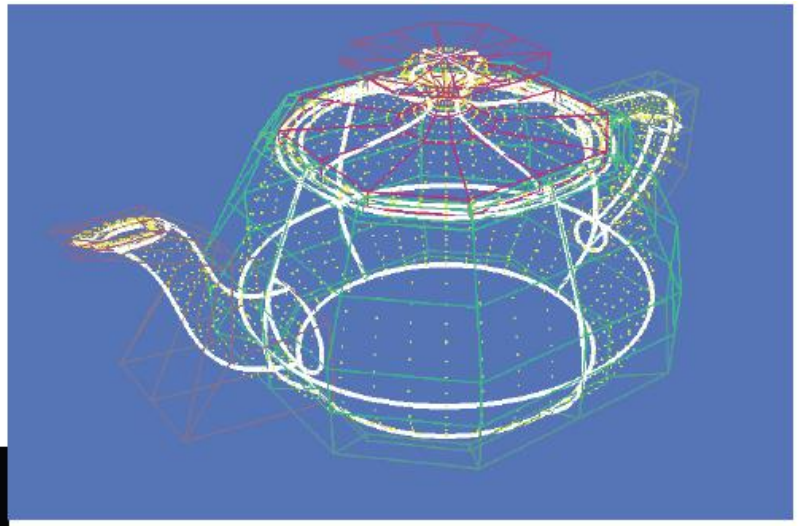
$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$



Бикубическая поверхность Безье: сопряжение



Бикубическая поверхность Безье: пример

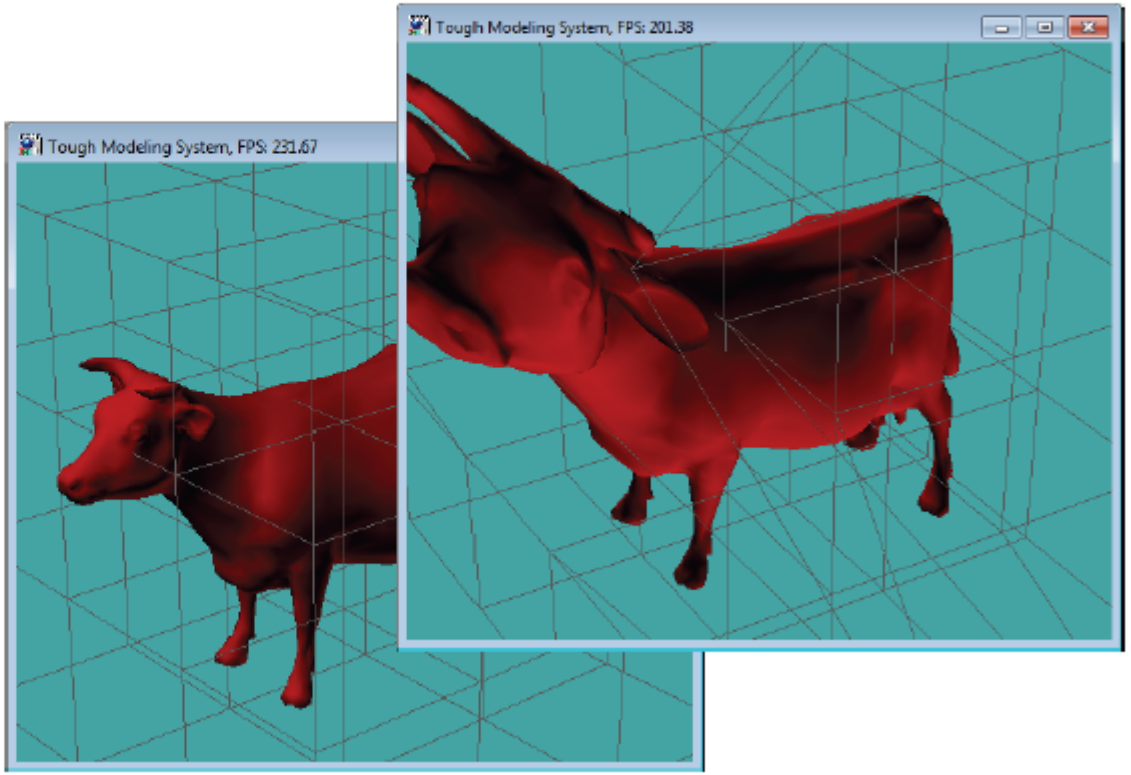


Трикубические пространства: Free form deformation

$$B(u,v,w)=\sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{l}P_{ijk}\cdot\mathbf{b}_{k,l}(w)\cdot\mathbf{b}_{j,m}(u)\cdot\mathbf{b}_{i,n}(v),\quad w,u,v\in[0,1]$$

$$\mathbf{b}_{i,n}(t)=C_n^i\cdot t^i\cdot(1-t)^{n-i}$$

$$C_n^i=\binom{n}{i}=\frac{n!}{i!(n-i)!}$$



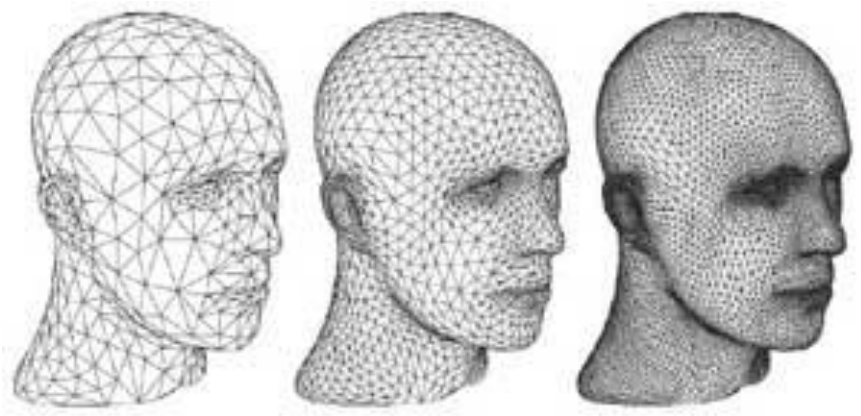
https://www.youtube.com/watch?v=H6KsxFAiPHc

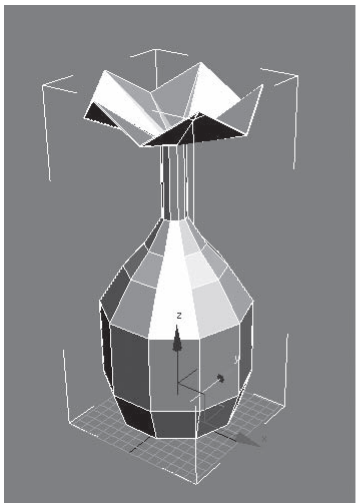
Представление объектов с помощью параметрического задания. Где используются

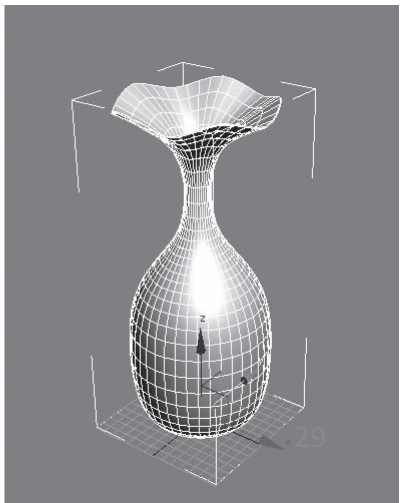
- Параметрические поверхности очень широко используются в CAD - системах (в России используется термин САПР - системы автоматического проектирования).
- Так, при помощи них моделируются и рассчитываются обводы автомобилей, формы деталей и т.п.

1.3. Представление пространственных форм с помощью полигональных сеток

- ***Полигональной сеткой (ПС)*** является совокупность связанных между собой плоских многоугольников.
- Наружная форма зданий, различные объекты, ограниченные криволинейными поверхностями.
- Недостатком метода является ***его приблизительность***. Хотя ***ошибки*** можно сделать ***сколь угодно малыми***, используя большее число многоугольников, но это приводит к дополнительным затратам ***памяти*** и ***вычислительного времени***.



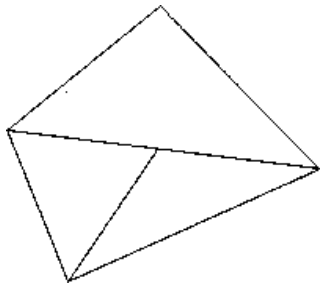




1.3. Представление пространственных форм с помощью полигональных сеток

Часто в качестве многоугольников используются треугольники, тогда разбиение поверхности называют триангуляцией. Топология полученной при этом сетки описывается следующим образом:

1. Объектами сетки являются вершины, ребра и треугольники (задаваемые тремя вершинами или тремя ребрами).
2. У любой вершины есть свойство валентности (т.е. число многоугольников, содержащих ее).
3. Для любого треугольника существует не более одного другого треугольника, инцидентного для первого по фиксированному ребру. Т.е., в частности, не может быть вот такой ситуации:



Нарушение условия об инцидентности

При таком способе задания мы можем, например, организовать обход в некотором направлении по треугольникам, содержащим фиксированную вершину, просто переходя каждый раз к треугольнику, инцидентному по ребру для предыдущего.

Способы задания многоугольников(1)

1. Явное задание многоугольников

- Каждый многоугольник можно представить в виде списка координат его вершин: $P = ((x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n))$
- Недостаток - дублирование информации о вершинах соседних многоугольников.

2. Задание многоугольников с помощью указателя на список вершин: $V = ((x_1, y_1, z_1), \ldots, (x_n, y_n, z_n))$.

Многоугольник, составленных из вершин 3,5,7,10 представляется как $P = (3, 5, 7, 10)$.

Недостаток- сложно отыскать многоугольники с общими ребрами.

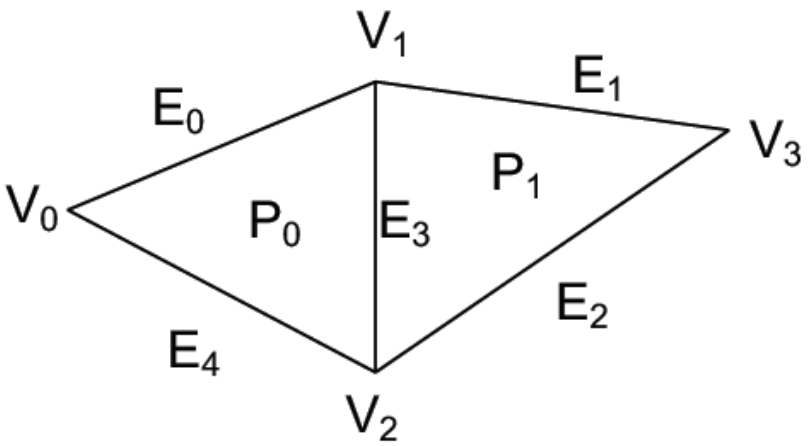
Способы задания многоугольников(2)

3. Явное задание ребер

Многоугольник - совокупность указателей на элементы списка ребер:

многоугольник как $P = (E_1, \ldots, E_2)$,

ребро — как $E = (V_1, V_2, P_1, P_2)$.



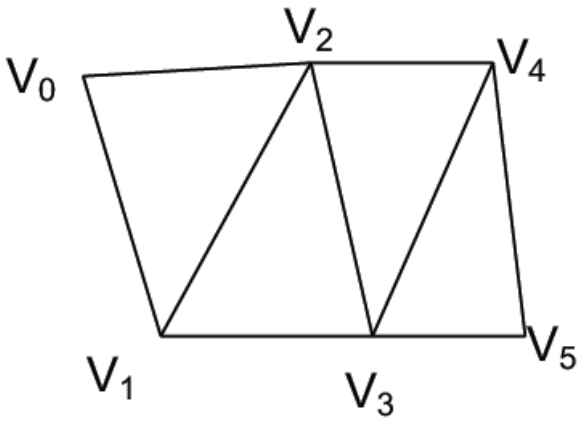
$V = \{V_0, V_1, V_2, V_3\}$

$E_0 = \{V_0, V_1, P_0, 0\}$
$E_1 = \{V_0, V_1, 0, P_1\}$
$E_2 = \{V_1, V_3, 0, P_1\}$
$E_3 = \{V_2, V_1, P_0, P_1\}$
$E_4 = \{V_0, V_2, P_0, 0\}$

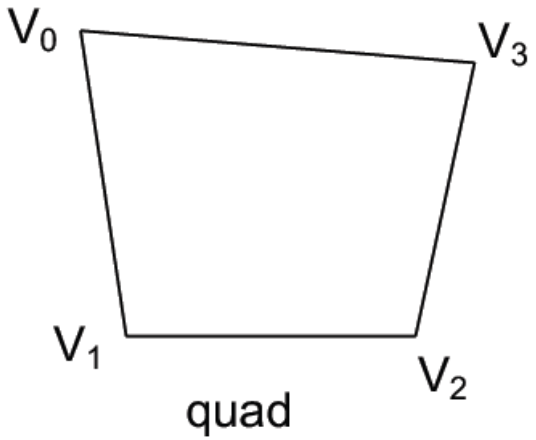
$P_0 = \{E_0, E_3, E_4\}$
$P_1 = \{E_1, E_2, E_3\}$

Полигональная сетка изображается путем вычерчивания всех ребер.

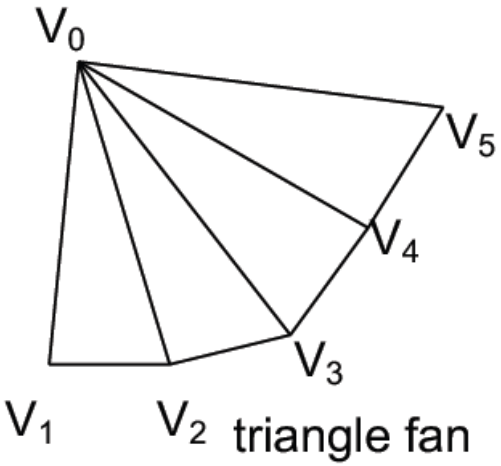
Представление объектов



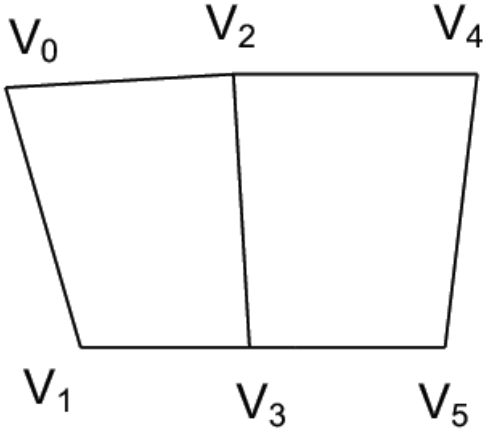
triangle strip
(0,1,2),(2,1,3),(2,3,4),(4,3,5)



quad



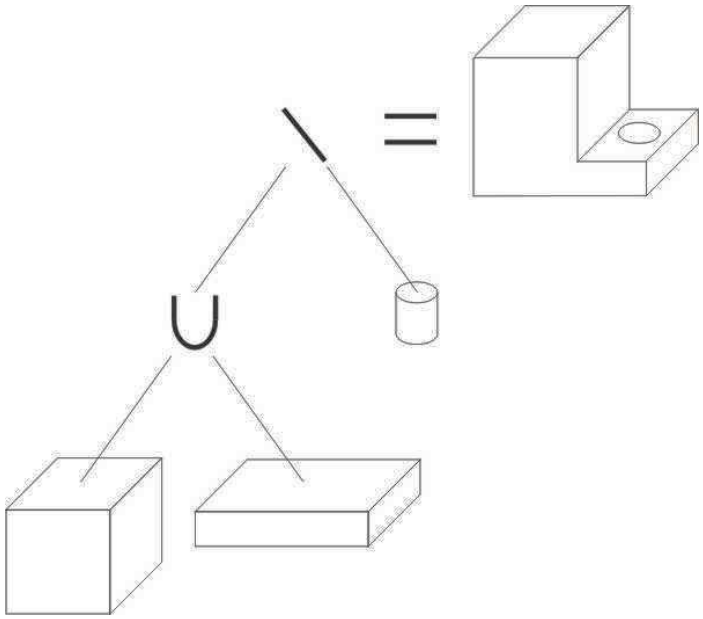
triangle fan
(0,1,2),(0,2,3),(0,3,4),(0,4,5)



quad strip

1.4. Конструктивная геометрия тел

- Нужная поверхность получается в результате применения последовательности различных множественных операций (***объединения, пересечения, разности*** и т.д.) к некоторым ***примитивам***.
- В качестве примитивов могу выступать ***параллелепипеды***, ***шары***, ***конусы***, ***пирамиды*** и т.д. Сама поверхность задается при помощи дерева построения.

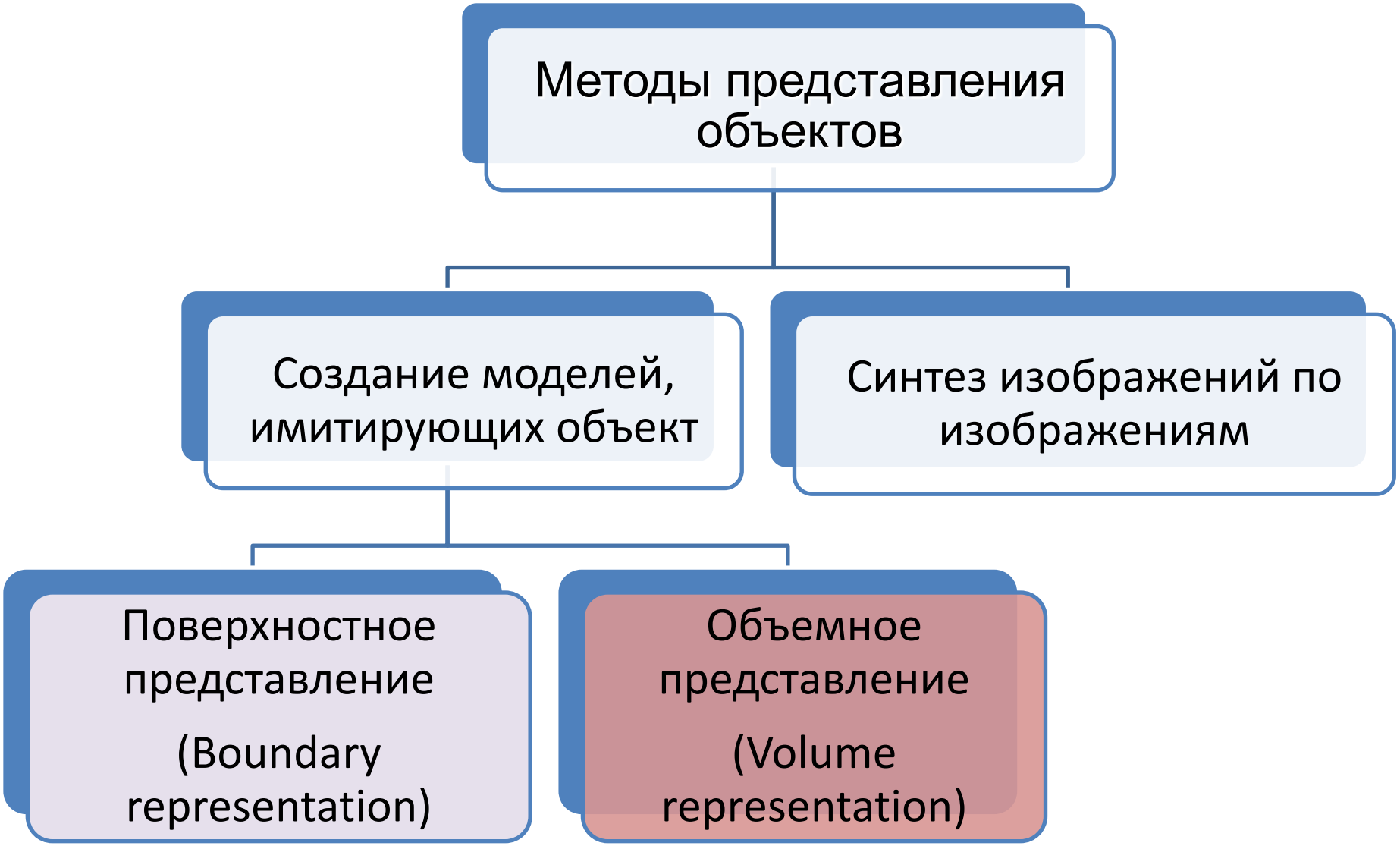


Пример дерева построения

Сопоставление методов представления пространственных форм

- Для представления поверхности с заданной точностью требуется значительно меньшее число бикубических кусков, чем при аппроксимации полигональной сеткой.

- Однако алгоритмы для работы с бикубическими объектами существенно сложнее алгоритмов, имеющих дело с многоугольниками.


Методы представления объектов
Создание моделей, имитирующих объект
Синтез изображений по изображениям
Поверхностное представление
(Boundary representation)
Объемное представление
(Volume representation)

2. Объемное представление (Volume representation)

Объемное представление

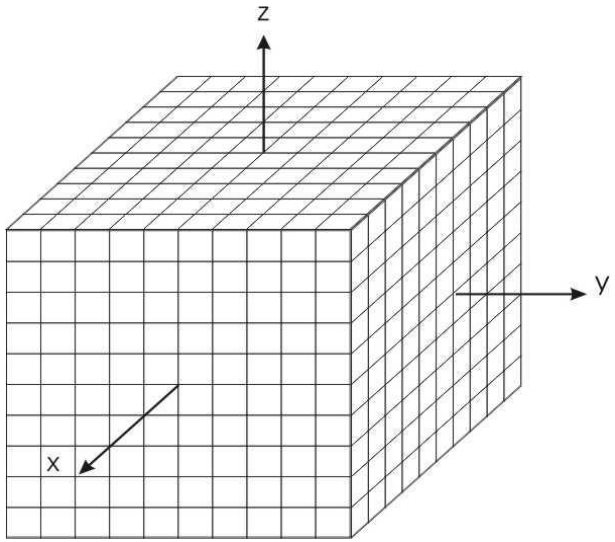
- Сохраняют информацию о любых, не обязательно видимых, частях объекта.
- Как правило, требуют гораздо больше места для хранения объекта, но дают дополнительную информацию.
- Эти методы необходимы, когда тела не имеют как таковой границы (например, туман, взвесь в жидкости и т.д.)

2.1. 3D растр

- 3D растр представляет собой ***набор кубиков*** (voxel'ов, от «volume element») в пространстве. Каждый воксел может иметь некоторое числовое значение, являющееся атрибутом соответствующей точки в пространстве.
- Данный метод, несмотря на свою простоту, имеет серьезный недостаток: для ***хранения*** даже небольшого объекта в приемлемом качестве ***требуется очень много места.*** Например, если мы храним кубик с ребром 1024 и выделяем по одному байту (8 бит) на атрибут воксела, то нам потребуется

$$1024 \times 1024 \times 1024 \times 1 байт = 1 Гб.$$

Учитывая то, что хранить объект надо будет в оперативной памяти компьютера, применение этого метода в чистом виде становится крайне затруднительным.



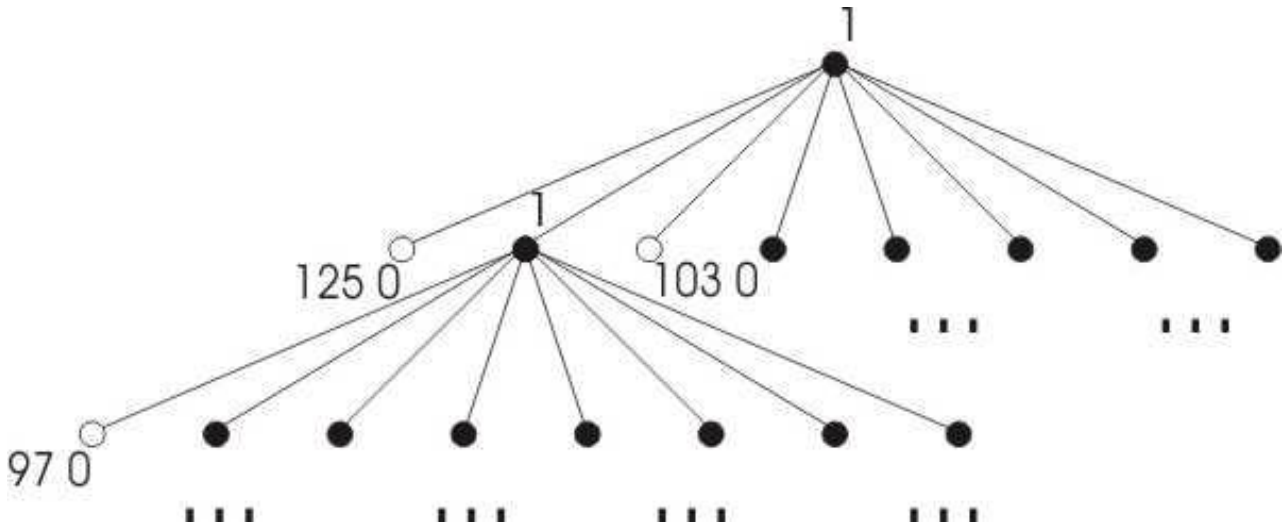
3D растр

2.2. Восьмеричное дерево

- В большинстве случаев в 3D растре есть области вокселов, имеющих ***одинаковые атрибуты***. Учитывая это, можно ***сократить объем*** занимаемого места.
- Будем хранить растр в виде ***дерева***, ***вершины*** которого ***имеют степень 0 или 8***. Сначала проверим, не лежат ли во всех вокселах в растре одинаковые значения, если да, то нам достаточно будет сохранить это значение (*p)* и обозначить, что дальше дробить кубик не надо, например, положим в вершину это значение *p* и 0 (обозначение терминальности вершины).
- Если у нас есть вокселы с разными значениями, то продолжим процесс. А именно, разделим растр на октанты, соответствующие координатным плоскостям.
  - Для каждого октанта снова проверим, не содержит ли он одинаковые вокселы. И т.д.

  Получим дерево, в вершинах которого лежат 1 (что обозначает, что соответствующий октант не однороден) или 0 и некоторые числа p_i, соответствующие значениям всех вокселов в данном октанте.

2.2. Восьмеричное дерево



Восьмеричное дерево 3D растра

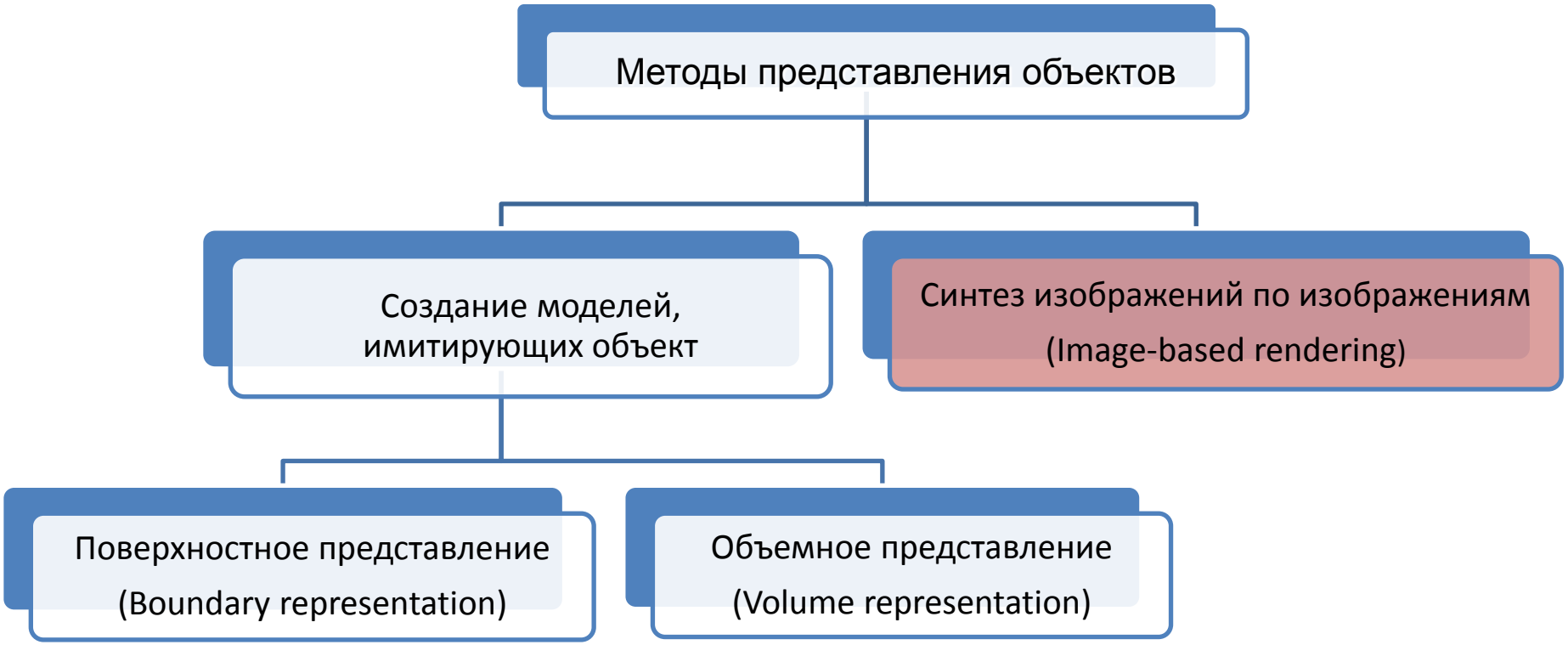
Таким образом, если растр состоит из вокселов, имеющих один и тот же атрибут, то мы сэкономим $1\,Гб - (1байт + 1бит)$, если же все воксели различны, то лишнее место, которое мы потратим (на 1 и 0), составит

$$(1 + 8 + 8^2 + ... + 8^{10})\ битов = 1227133513\ битов \approx 146\,мб$$

Но такая ситуация встречается крайне редко, поэтому использование восьмеричных деревьев в большинстве случаев позволяет значительно сократить объем памяти для хранения объекта.

2.3. Двоичное дерево

- Идея построения двоичного дерева полностью аналогична построению восьмеричного дерева.
- Но в данном случае растр последовательно делится не на октанты, а на половинки: сначала параллельно плоскости OXY, потом OYZ и т.д. Соответственно, аналогично строится дерево, вершины которого имеют степень 0 или 2.



Метод синтеза изображений по изображениям стоит несколько особняком, так как не позволяет осуществлять никаких действий, кроме визуализации объектов.

Синтез изображений (Image-based rendering, immersive imaging)

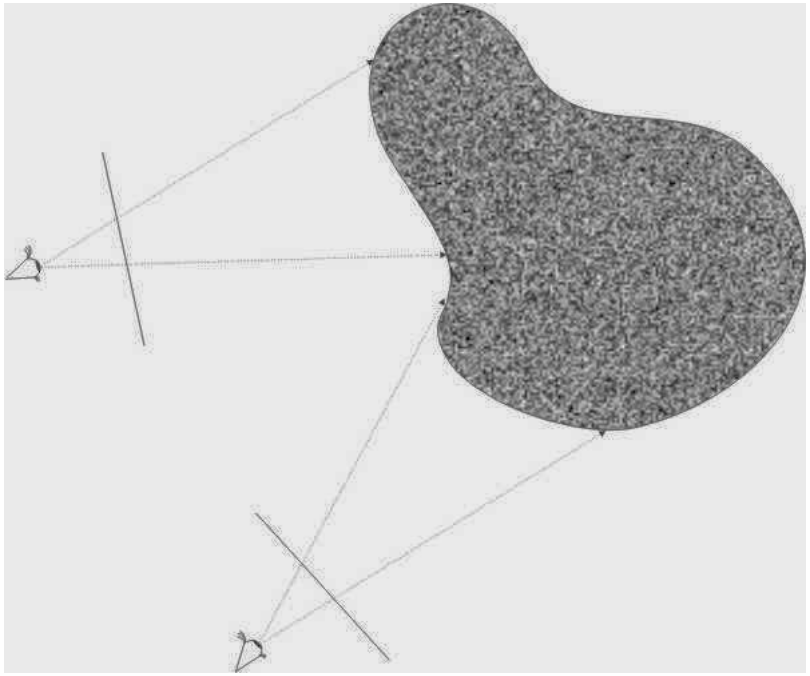
Синтез изображений

- Идея состоит генерации изображения трёхмерного объекта на мониторе по набору существующих 2D изображений (например, фотографий).
- При этом нам не приходится решать трудоемкую задачу создания моделей объектов.

Синтез изображений. Общая идея

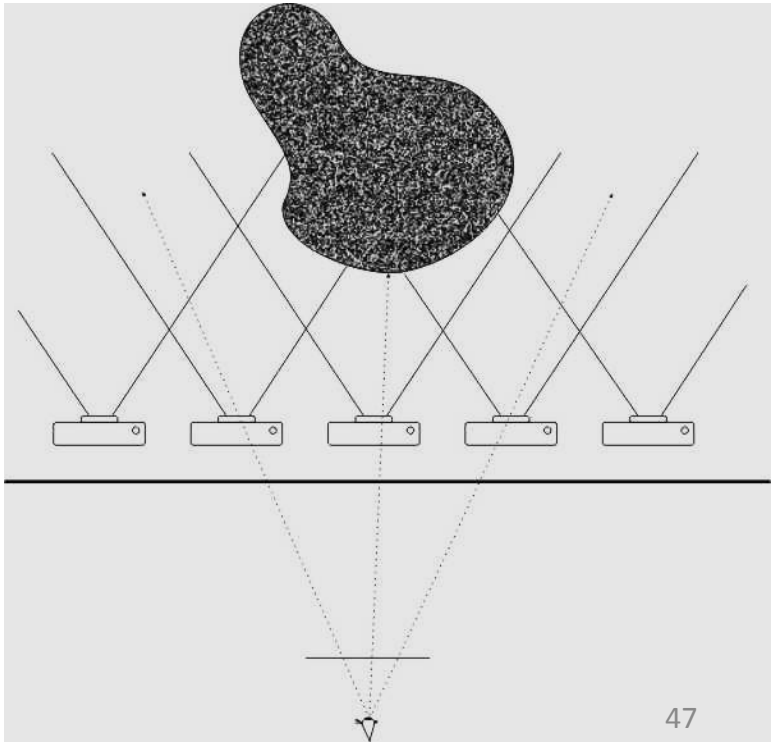
- Мы хотим генерировать изображение трехмерной сцены по набору изображений. На эту задачу можно посмотреть как на нахождение значений некоторой функции (т.е. атрибутов пикселей) от 5 параметров: x, y, z, φ и θ,
- где x, y, z - координаты наблюдателя, а φ и θ - углы, задающие направление луча зрения.

Размерность массива данных, которые нам потребуются при таком подходе, составит, таким образом, 5D.



Синтез изображений. Lumigraph

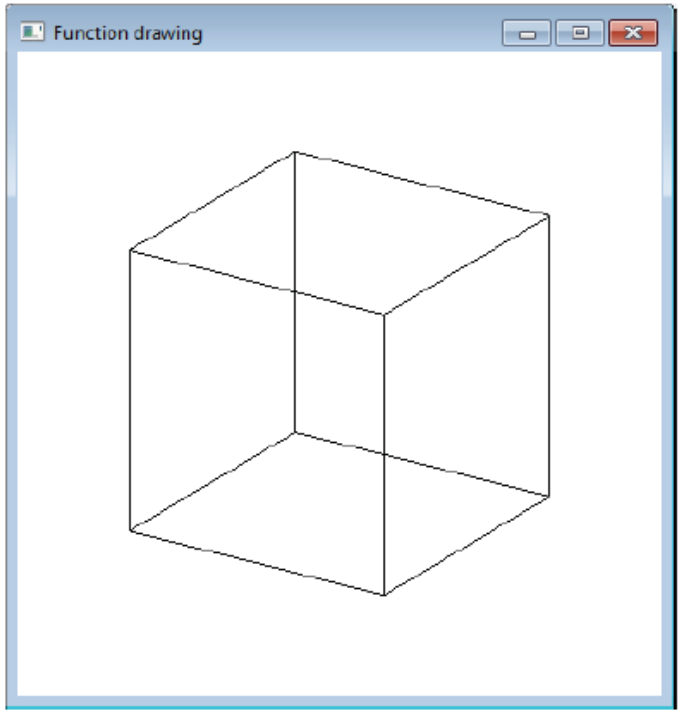
- Если мы знаем, что наблюдатель не может находиться за некоторой линией, можно сократить объем данных: сфотографировать объект вдоль этой линии (на всех уровнях по вертикали), а затем в качестве значений пикселя, соответствующего некоторому лучу зрения, брать значение, соответствующее ближайшему лучу камеры.

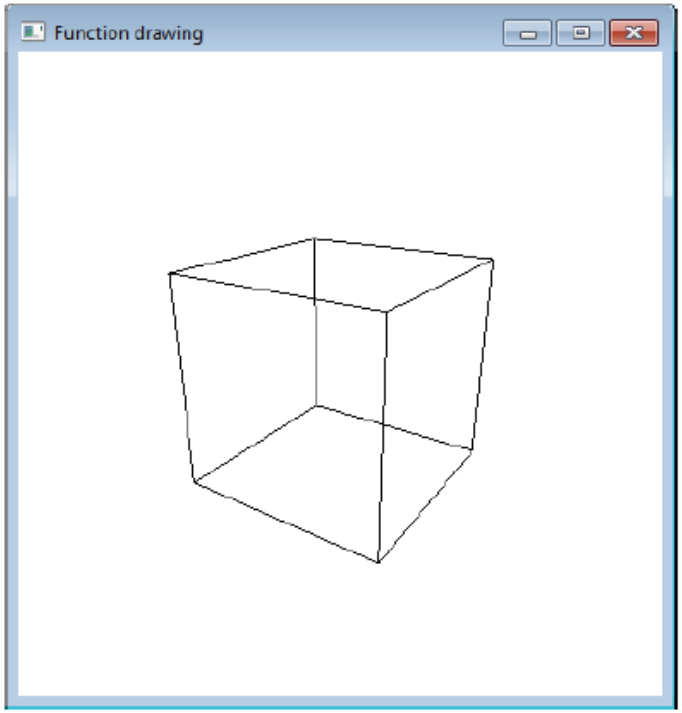


Размерность массива данных
в этом случае будет составлять 4:
2 на положение камеры
2 на изображение от каждой камеры).

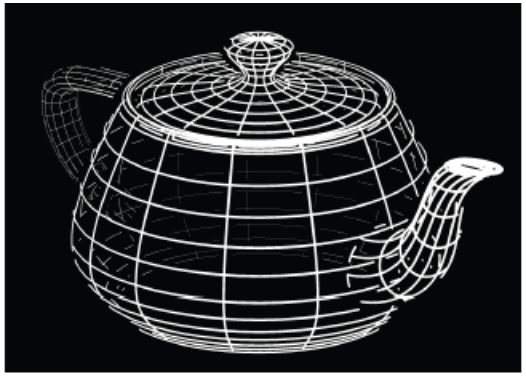
Визуальный реализм

- Визуальный реализм
  - перспектива:

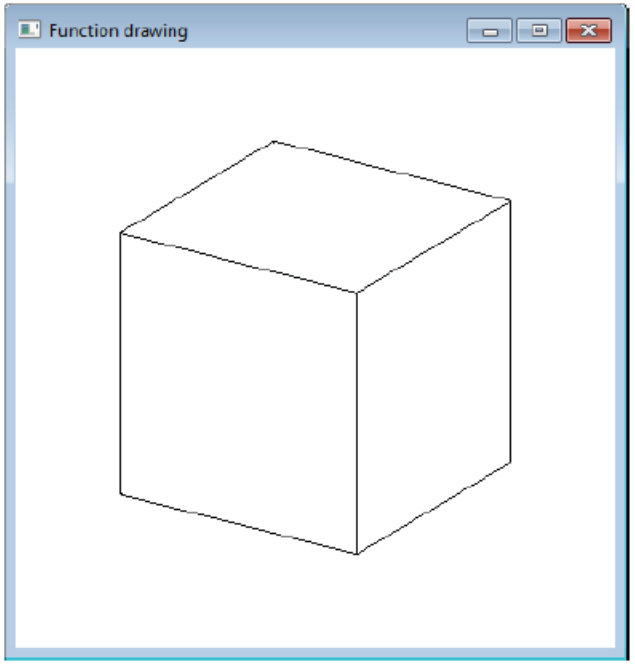


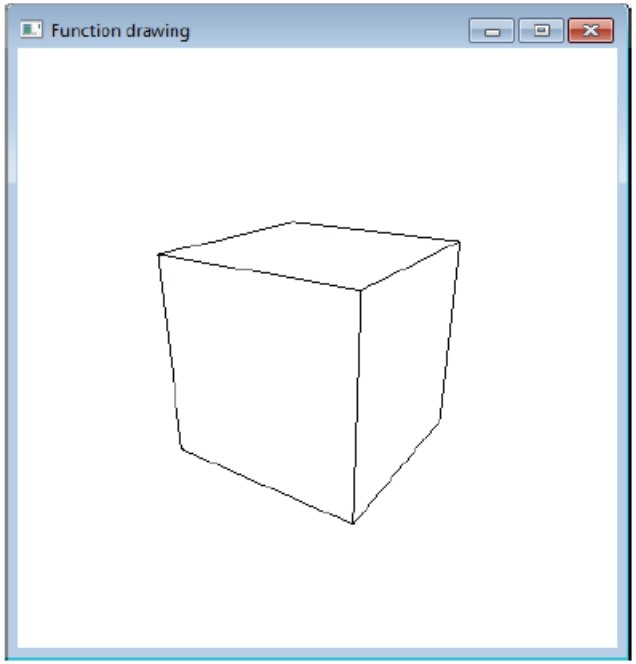


- Визуальный реализм
  - depth cueing (изображение глубины):

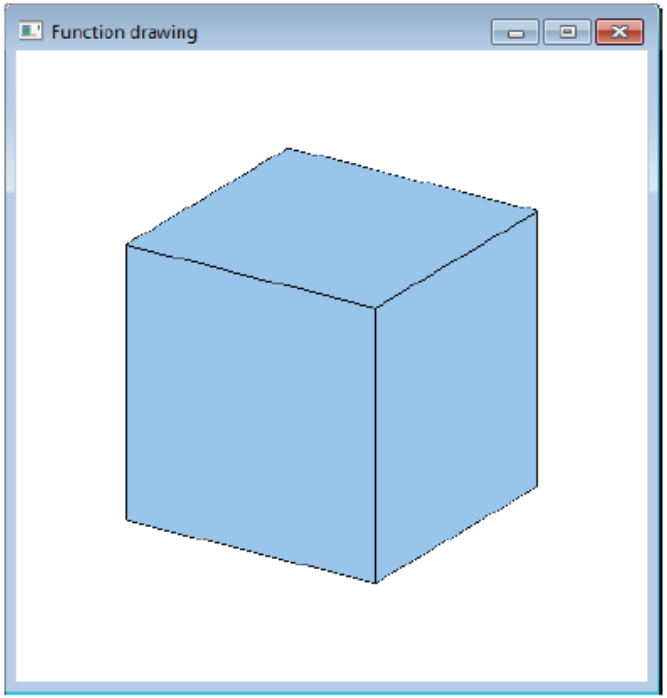


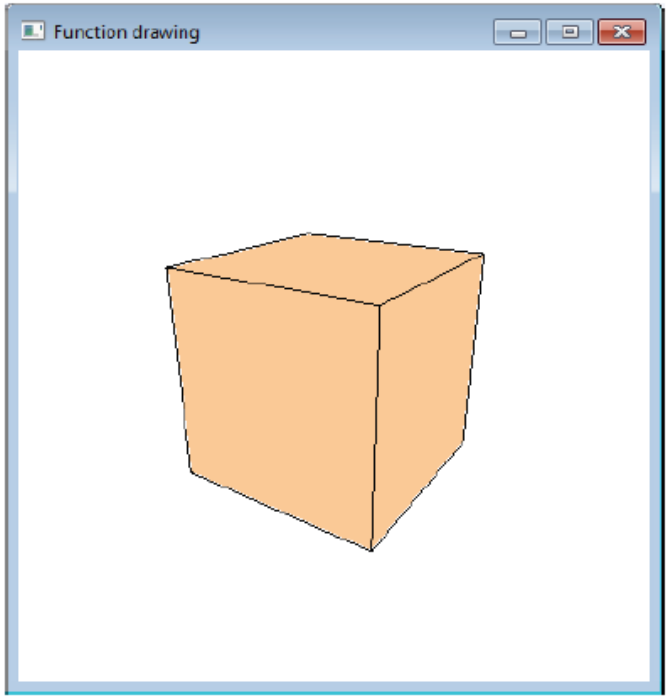
  - удаление невидимых линий



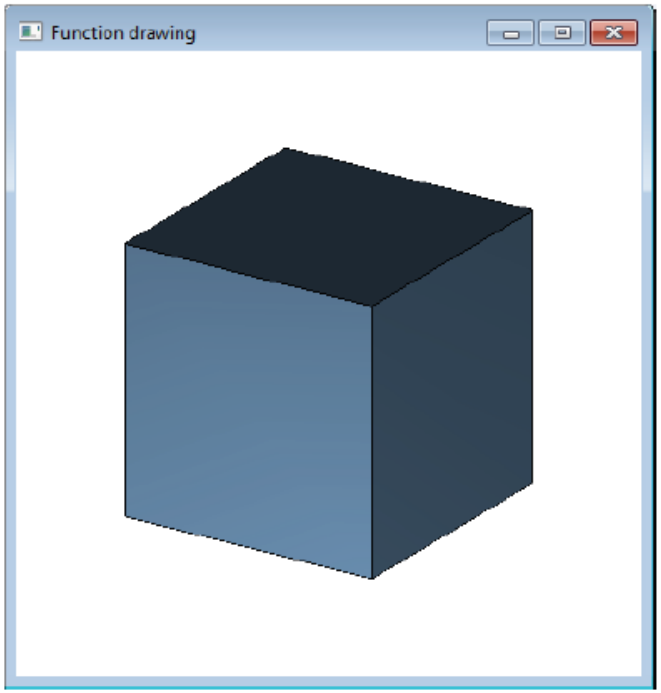


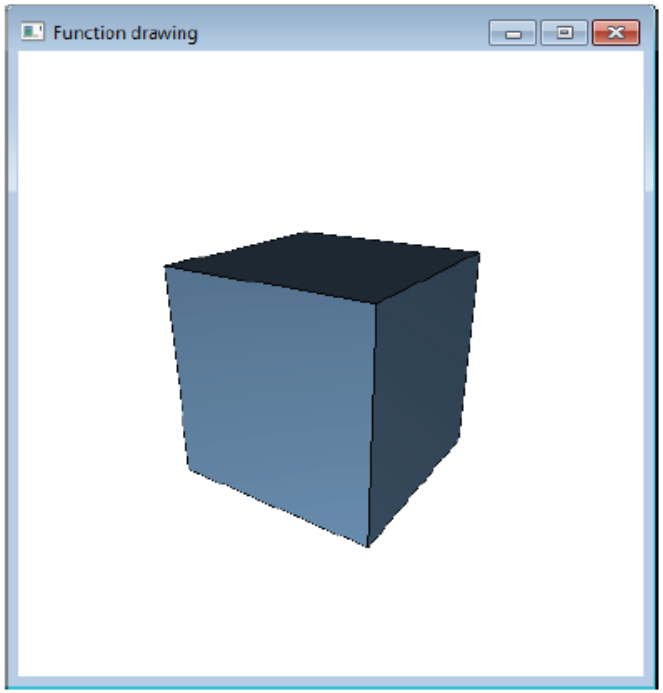
- Визуальный реализм
  - удаление невидимых поверхностей
  - цвет



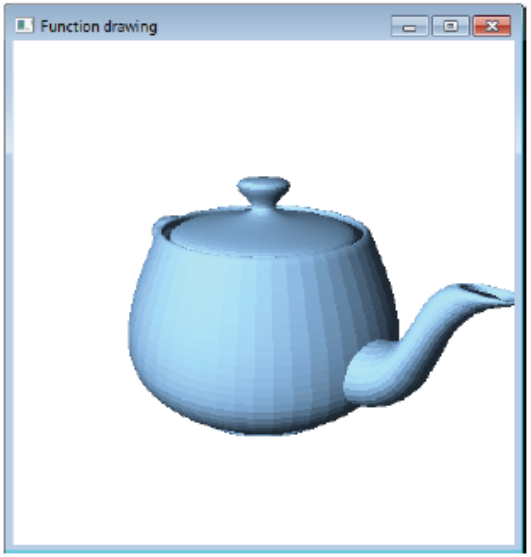


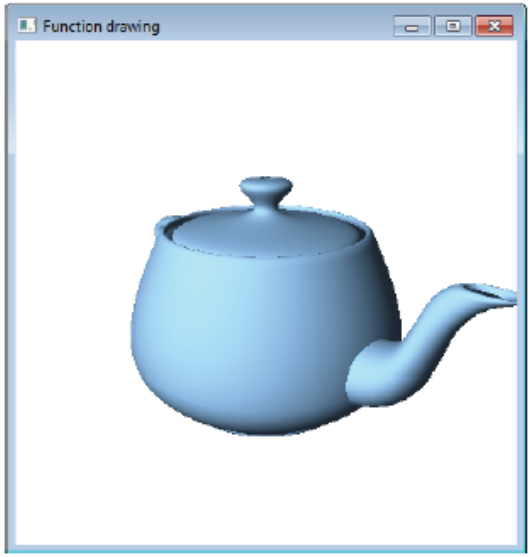
- Визуальный реализм
  - освещение



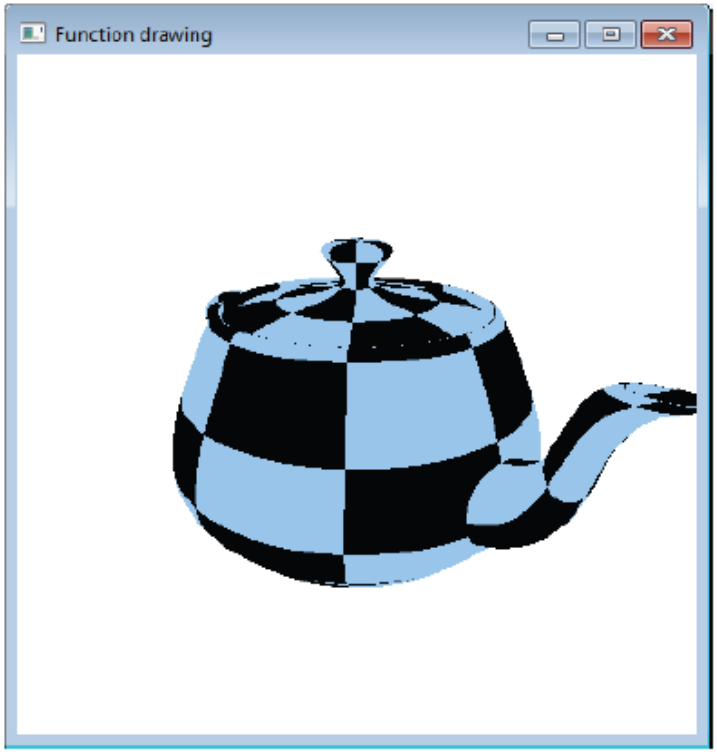


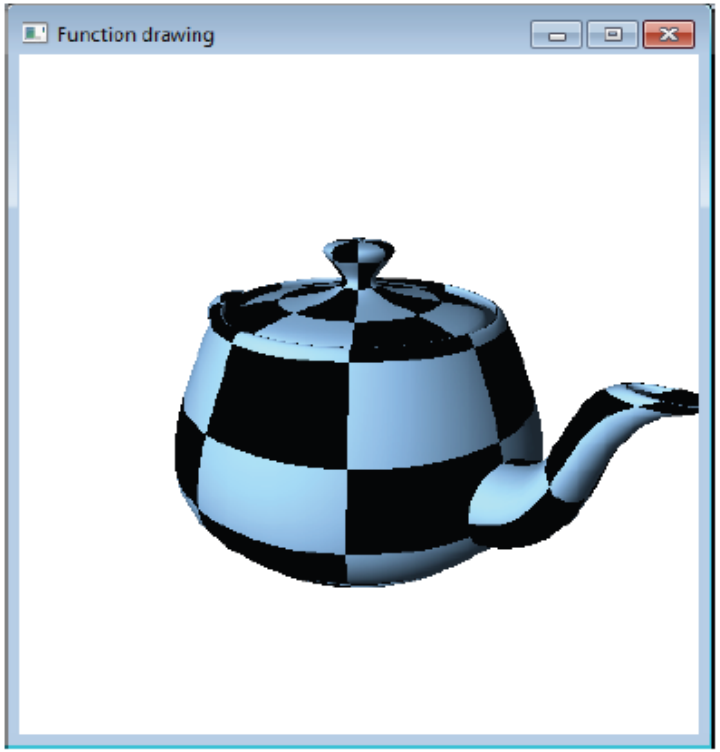
- Визуальный реализм
  - закраска и интерполяция



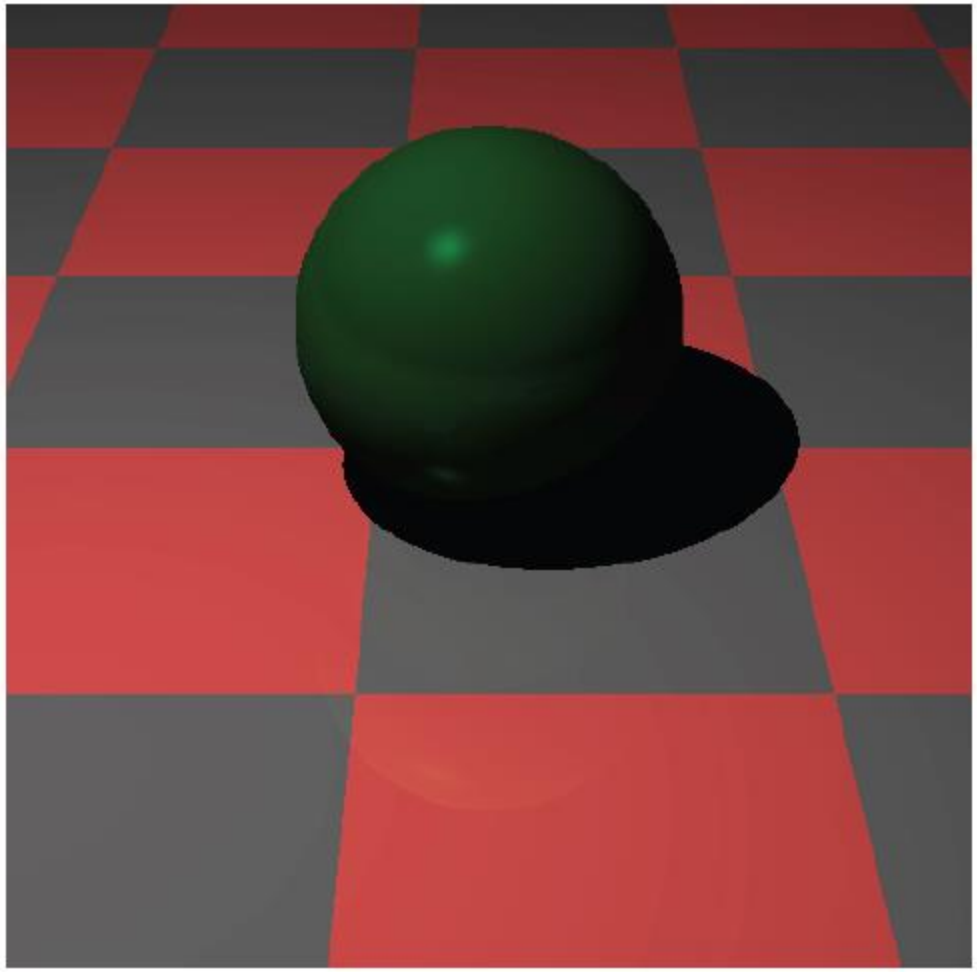


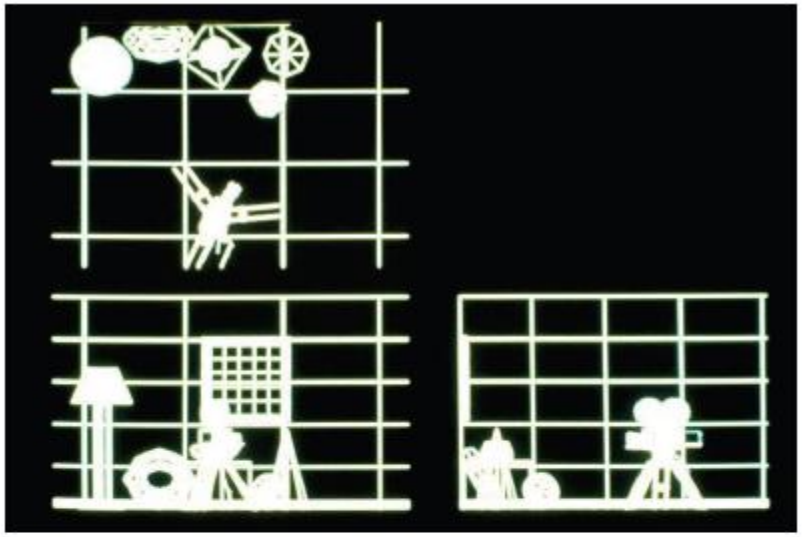
- Визуальный реализм
  - текстурирование



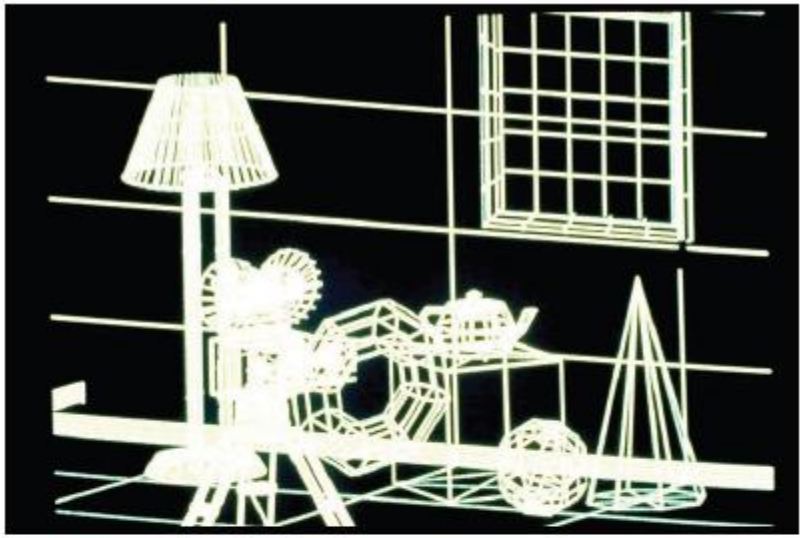


- Визуальный реализм
  - тени



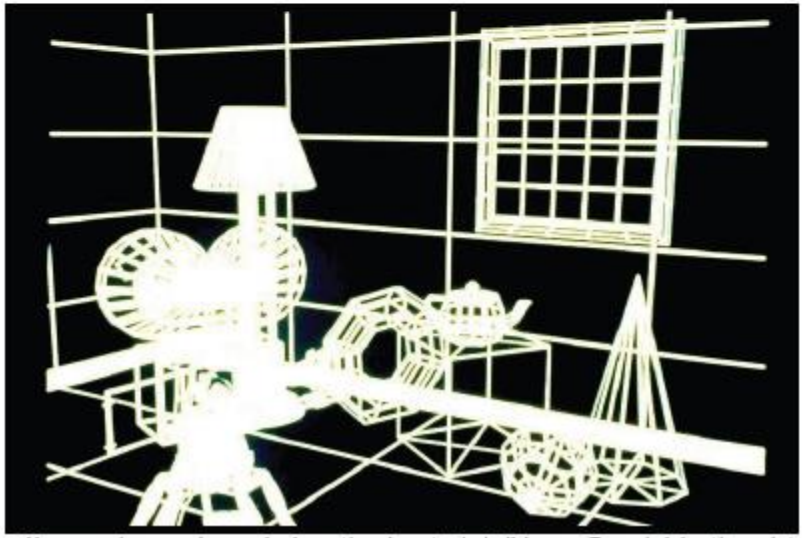


3 Orthographic views

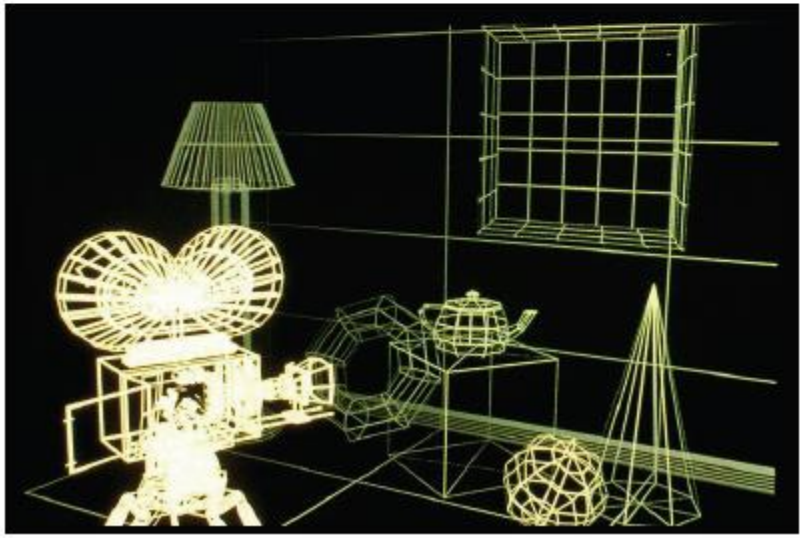


Parallel projection

Perspective View (no hidden lines)



Depth cuing (hidden lines)

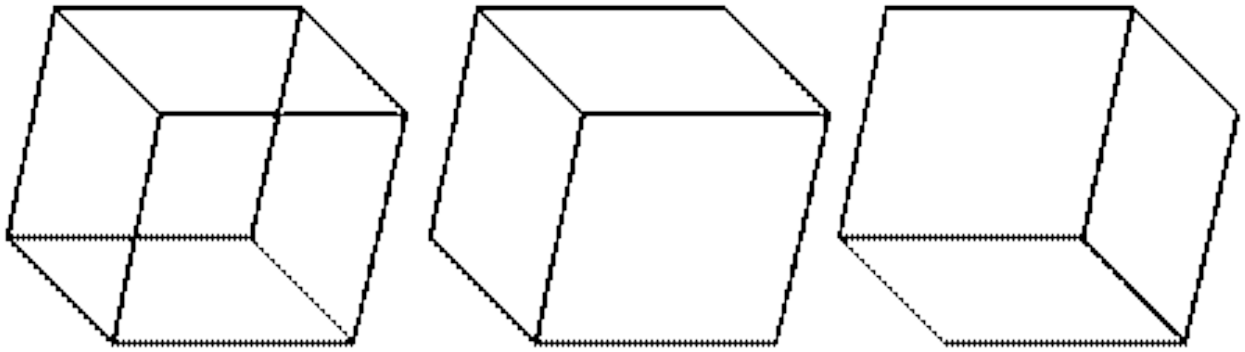


http://www.siggraph.org/education/materials/HyperGraph/shuthug.htm

Удаление невидимых линий и поверхностей

Необходимость удаления невидимых линий

- Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике.
- Алгоритмы удаления невидимых линий и поверхностей служат для ***определения линий ребер, поверхностей или объемов***, которые ***видимы*** или ***невидимы*** для наблюдателя, находящегося в заданной точке пространства.



Необходимость удаления невидимых линий

Удаления невидимых линий и поверхностей

- В общем случае решаются две задачи:
  - удалить все нелицевые поверхности;
  - удалить все видимые поверхности, закрываемые другими объектами (если объектов несколько).

Удаления невидимых линий и поверхностей

1. Определить: какие рёбра и грани видимы у конкретного объекта.
2. Если объектов на сцене несколько, то требуется определить: какие части объектов закрываются другими объектами.

Оптимального решения в общем виде для данной задачи не существует.

Удаления невидимых линий и поверхностей

Большинство алгоритмов удаления невидимых линий и поверхностей используют алгоритмы ***сортировки*** и ***когерентности***.

- ***Сортировка*** облегчает сравнение по глубине.
- ***Когерентность*** учитывает регулярность структур сцены.

Удаления невидимых линий и поверхностей

- **Сортировка ведется по геометрическому расстоянию от тела**, поверхности, ребра или точки до точки наблюдения.
- Чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен одним из объектов, более близких к точке наблюдения.

- После определения расстояний или приоритетов по глубине проводят **сортировку по горизонтали и по вертикали**, чтобы выяснить, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения.

Классификация алгоритмов удаления невидимых линий и поверхностей (1)

- **Алгоритмы, работающие в объектном пространстве**, имеют дело с физической системой координат, в которой они описаны (3D) (точные результаты). Полученные изображения можно свободно масштабировать.
- **Алгоритмы, работающие в пространстве изображения**, имеют дело с системой координат того экрана, на котором объекты визуализируются (2D) (точность вычислений ограничена разрешающей способностью экрана).
- Алгоритмы, формирующие список приоритетов, работают попеременно в обеих упомянутых системах координат.

Классификация алгоритмов удаления невидимых линий и поверхностей (2)

- **Алгоритмы, работающие в объектном пространстве.**
  - Алгоритм Робертса.
- **Алгоритмы, работающие в пространстве изображения (экрана).**
  - Алгоритм плавающего горизонта;
  - Алгоритм с использованием *z-буфера*;
  - Метод трассировки лучей;
  - Алгоритм Варнока;
  - Алгоритм Вейлера − Азертона.
- **Алгоритмы, формирующие список приоритетов.**
  - Алгоритм художника;
  - Алгоритм *Ньюэла – Ньюэла – Санча* (алгоритм удаления невидимых граней методом сортировки по глубине).

Классификация алгоритмов удаления невидимых линий и поверхностей (3)

- Объем вычислений для алгоритма, работающего в объектном пространстве растет, как квадрат числа объектов (n^2).
- Объем вычислений любого алгоритма, работающего в пространстве изображения и сравнивающего каждый объект сцены с позициями всех пикселов в системе координат экрана, растет теоретически, как $n*N$, где n - количество объектов (тел, плоскостей или ребер) в сцене, а N — число пикселей.
- **Теоретически** трудоемкость алгоритмов, работающих в объектном пространстве, меньше трудоемкости алгоритмов, работающих в пространстве изображения, при $n < N$. Однако на **практике** алгоритмы, работающие в пространстве изображения, более эффективны потому, что для них легче воспользоваться преимуществом когерентности при растровой реализации.

Алгоритм Робертса

Алгоритм Робертса (1963)

- Первое решение задачи об удалении невидимых линий для векторных устройств вывода. Метод работает в **объектном пространстве**.
- Требуется, чтобы моделируемое тело было выпуклым. Невыпуклые объекты должны быть разделены на выпуклые.
- **Суть алгоритма:** на основании расчета нормалей к каждой грани объекта, все грани, имеющие неположительные нормали по отношению линии наблюдения, полагаются нелицевыми. Разделение граней объекта на лицевые и нелицевые позволяет решить вопрос о видимости ребер.

Алгоритм Робертса

Алгоритм выполняется в 3 этапа:

- **На 1-м этапе** объекты анализируются индивидуально с целью удаления нелицевых плоскостей.
- **На 2-м этапе** проверяется экранирование оставшихся в каждом теле рёбер всеми другими телами с целью обнаружения их невидимых отрезков.
- **На 3-м этапе** вычисляются отрезки, которые образуют новые рёбра при протыкании объектов друг друга.

В алгоритме предполагается, что объекты состоят из плоских полигональных граней, которые в свою очередь ограничены рёбрами, а рёбра – отдельными вершинами. Все вершины, рёбра и грани связаны с конкретным объектом.

Алгоритм Робертса

Первый этап (удаление невидимых граней):

- Для каждого объекта сцены:
  - Сформировать многоугольники граней, рёбра на основе списка вершин объекта.
  - Вычислить уравнение плоскости для каждой грани объекта.
  - Получить матрицу граней объекта.
  - Произвести её проективное преобразование.
  - Вычислить и запомнить габариты описывающего проекцию объекта прямоугольника: Xmax, Xmin, Ymax, Ymin
  - Определить нелицевые плоскости:
    - Вычислить скалярное произведение вектора, направленного к наблюдателю, на грань из матрицы объекта.
    - Если скалярное произведение меньше нуля – грань невидима.
    - Удалить весь описывающий многоугольник данной грани.
- Если в сцене лишь 1 объект – алгоритм завершён, иначе – 2-й этап.

Алгоритм Робертса

Второй этап (проверка экранирования объектов друг другом):

- Сформировать список «приоритета» объектов, упорядоченных по Z координате. Наиболее удалённый объект будет стоять первым в списке.
- Для каждого тела из списка:
  - Проверить экранирование всех лицевых рёбер объекта всеми другими телами сцены, стоящими ниже в списке.
  - Проверка производится сперва по описывающим проекции граням прямоугольникам.
  - Если прямоугольники не пересекаются (в плоскости XY), то переход к следующему объекту.
  - Иначе проверка на протыкание.

Алгоритм Робертса

2-й этап (продолжение):

- Проверяется возможность частичного экранирования.

- Если ребро полностью видимо-> далее. Иначе вычисляются видимые участки рёбер (сохраняются отдельно для их проверки с объектами более низкого приоритета).

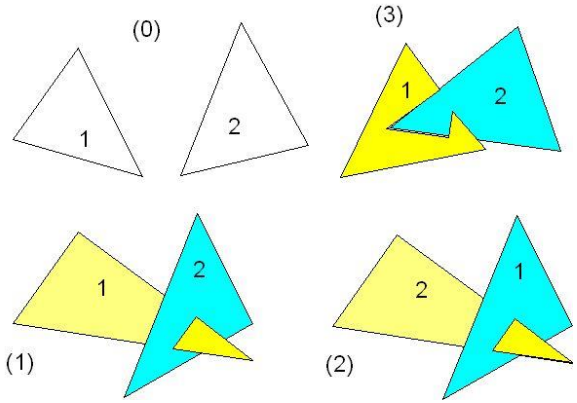
- Возможны случаи:
  - –грань не закрывает ребро;
  - –грань полностью закрывает ребро;
  - –грань частично закрывает ребро (в этом случае ребро разбивается на несколько частей, из которых видимыми являются не более двух).

Алгоритм Робертса

Проверка на протыкание:

Сравнить макс. значение Z проверяемого объекта с минимальным значением Z «экранируемого» объекта.

- Если Z_1max < Z_2min, то протыкание невозможно. Перейти к следующему экранируемому объекту. В противном случае - проверить видимое протыкание.
- Если условия протыкания выполнены – установить флаг протыкания (для активизации третьего шага) и занести оба объекта в список протыканий.



Алгоритм Робертса

3-й этап (протыкание):

- Сформировать все возможные рёбра, соединяющие точки протыкания, для пар объектов, связанных отношением протыкания.
- Проверить экранирование всех соединяющих рёбер гранями обоих объектов, связанных отношением протыкания.
- Проверить экранирование оставшихся соединяющих рёбер всеми прочими объектами сцены. Запомнить видимые отрезки.
- Визуализировать все видимые грани и отрезки рёбер.

Алгоритм Робертса

- Недостатки алгоритма Робертса :
  - неспособность без привлечения других подходов реализовать падающие тени;
  - невозможность передачи зеркальных эффектов и преломления света;
  - строгая ориентация метода только на выпуклые многогранники.
- Преимущество – одно: относительная простота.

Алгоритм Робертса: Определение нелицевых граней

Плоскость в 3D пространстве

- Уравнение произвольной плоскости в трехмерном пространстве:

$$ax + by + cz + d = 0$$

- Матричная форма плоскости:

$$[x\ y\ z\ 1][P]^T = 0,$$

где $[P]=[a\ b\ c\ d]$ – коэффициенты, описывающие плоскость.

- Точка $(x_1,y_1,z_1,1)$ расположена за плоскостью, если:

$$[x_1\ y_1\ z_1\ 1][P]^T < 0$$

Модель объекта-многогранника 3D

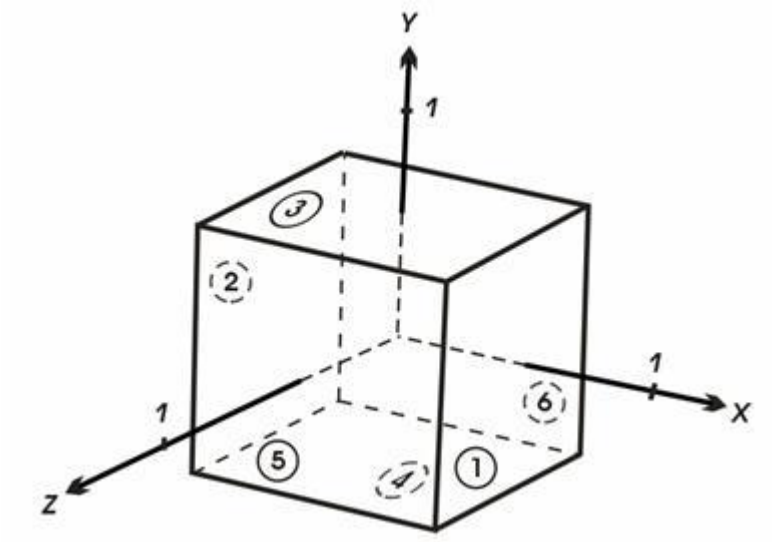
Любое выпуклое твердое тело можно выразить матрицей, состоящей из коэффициентов уравнений составляющих его плоскостей, т. е.

$$V = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{bmatrix},$$

где каждый столбец содержит коэффициенты одной плоскости.

Определение нелицевых граней

- Пусть *F* — некоторая грань многогранника. Плоскость, несущая эту грань, разделяет пространство на два подпространства. Назовем положительным то из них, в которое смотрит внешняя нормаль к грани.



Определение нелицевых граней

- Если многогранник выпуклый, то удаление всех нелицевых граней полностью решает задачу визуализации с удалением невидимых граней.
- Для определения, лежит ли точка в положительном подпространстве, используют проверку знака скалярного произведения (l, n)

где l – вектор, направленный к наблюдателю;

n – вектор внешней нормали грани.

- Если (l, n) > 0, т. е. угол между векторами острый, то грань является лицевой.
- Если (l, n) < 0, т. е. угол между векторами тупой, то грань является нелицевой.

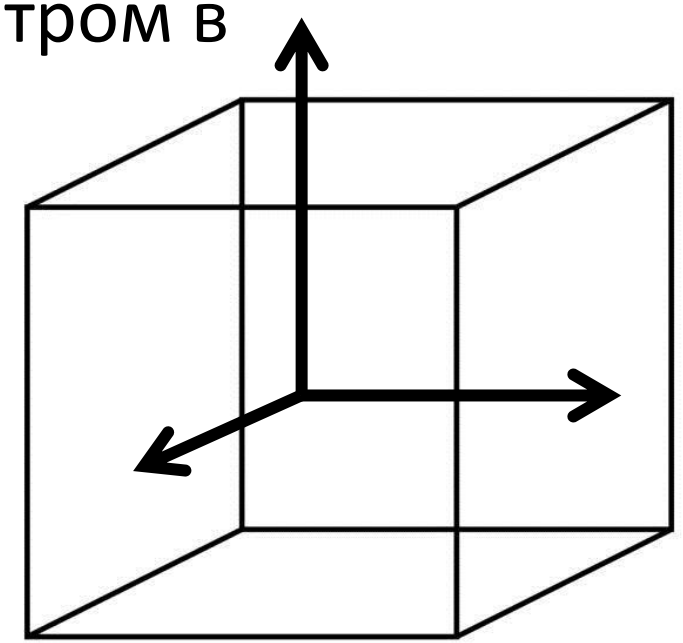
Задание граней

- Рассмотрим единичных куб с центром в начале координат.
- 6 плоскостей, описывающих куб:

x_1=1/2, x_2=-1/2

y_3=1/2, y_4=-1/2

z_5=1/2, z_6=-1/2



Полная матрица куба:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ -1/2 & 1/2 & -1/2 & 1/2 & -1/2 & 1/2 \end{bmatrix}$$

Проверка корректности граней

- Экспериментально проверим матрицу тела с помощью точки, лежит внутри тела.
- Если знак скалярного произведения для какой-нибудь плоскости больше нуля, то соответствующее уравнение плоскости следует умножить на -1.
- Возьмем точку внутри куба с координатами x = 1/4, y = 1/4, z = 1/4.
- В однородных координатах эта точка представляется в виде вектора:

$$[S] = [1/4 \;\; 1/4 \;\; 1/4 \;\; 1] = [1 \;\; 1 \;\; 1 \;\; 4].$$

Проверка корректности граней

- Скалярное произведение этого вектора на матрицу объема равно:

$$[S]\cdot[V] = [1\ \ 1\ \ 1\ \ 4]\cdot\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ -1/2 & 1/2 & -1/2 & 1/2 & -1/2 & 1/2 \end{bmatrix} = [-2\ \ 6\ \ -2\ \ 6\ \ -2\ \ 6].$$

- Корректная матрица равна:

$$\begin{bmatrix} 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Общий случай проверки

1. Хотя уравнение плоскости содержит четыре неизвестных коэффициента, его можно нормировать так, чтобы d = 1. Подстановка координат трех неколлинеарных точек плоскости (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3) в нормированное уравнение дает:

$$ax_1 + by_1 + cz_1 = -1;$$
$$ax_2 + by_2 + cz_3 = -1;$$
$$ax_3 + by_2 + cz_3 = -1.$$

- В матричной форме:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

или [X][C] = [D].
Решение этого уравнения дает значения коэффициентов уравнения плоскости: $[C] = [X]^{-1}[D]$.

Общий случай проверки (2)

2. Другой способ используется, если известен вектор нормали к плоскости, т. е.

$$\mathrm{n} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k},$$

где ***i, j, k*** – единичные векторы осей *x*, *y*, *z* соответственно. Тогда уравнение плоскости примет вид

$$ax + by + cz + d = 0$$

Величина *d* вычисляется с помощью произвольной точки на плоскости. В частности, если компоненты этой точки на плоскости (x_1, y_1, z_1), то

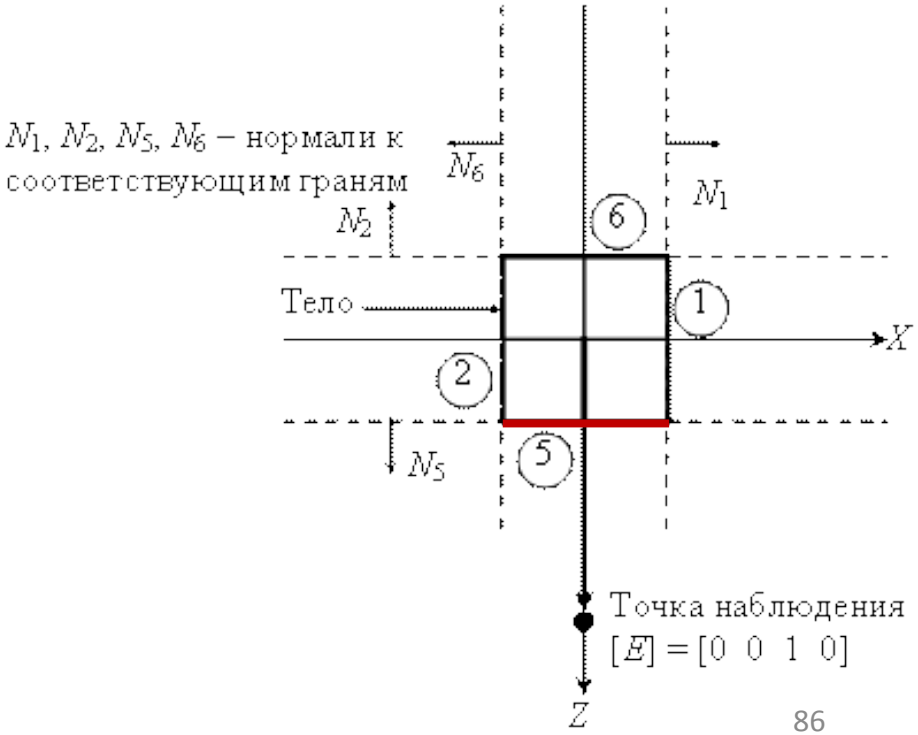
$$d = -(ax_1 + by_1 + cz_1).$$

Пример работы алгоритма Робертса. Параллельная проекция

Т.е. положительное скалярное произведение дает такая плоскость (столбец) в матрице тела, относительно которой точка лежит снаружи, т. е. в положительном подпространстве.

Рассмотрим пример:

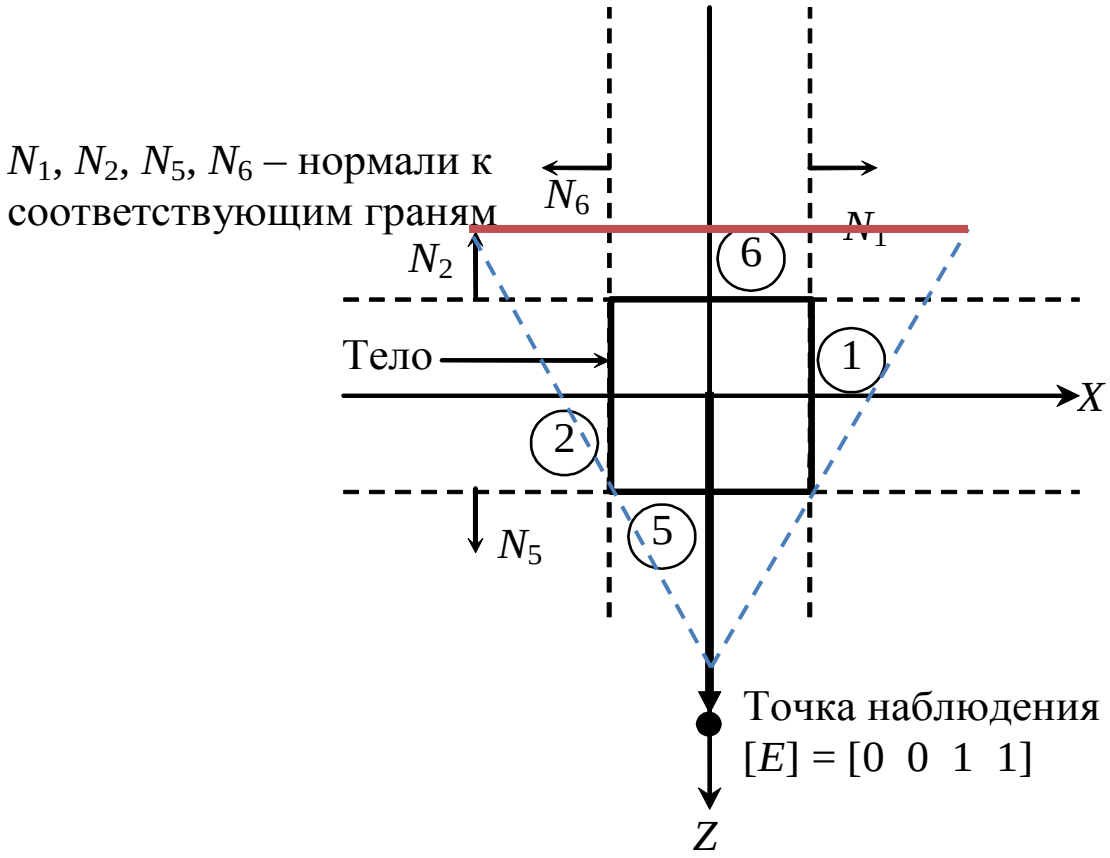
Условие $[E]\cdot[V] < 0$ определяет, что плоскости — нелицевые, а их грани — задние.



$$[E]\times[V] = [0\ \ 0\ \ 1\ \ 0] \times \begin{bmatrix} 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$= [0\ \ 0\ \ 0\ \ 0\ \ 2\ \ -2].$$

Пример работы алгоритма. Перспективная проекция



$$[E]\cdot[V] = [0\ \ 0\ \ 1\ \ 1]\cdot\begin{bmatrix} 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} = [-1\ \ -1\ \ -1\ \ -1\ \ 1\ \ -3].$$

Вычисление координат вектора нормали

- Определение вектора нормали к поверхности в заданной точке может быть выполнено различными способами.
- Для поверхностей, заданных в аналитической форме, известны методы дифференциальной геометрии, которые основываются на вычислении частных производных функций описания.

Вычисление координат вектора нормали

Если поверхность задана параметрическими функциями: $x = x(s,t),$

$y = y(s,t),$

$z = z(s,t),$

то координаты вектора нормали можно вычислить так:

$$x_N = \begin{vmatrix} \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{vmatrix} = \frac{\partial y}{\partial s} \cdot \frac{\partial z}{\partial t} - \frac{\partial y}{\partial t} \cdot \frac{\partial z}{\partial s},$$

$$y_N = \begin{vmatrix} \frac{\partial z}{\partial s} & \frac{\partial x}{\partial s} \\ \frac{\partial z}{\partial t} & \frac{\partial x}{\partial t} \end{vmatrix} = \frac{\partial z}{\partial s} \cdot \frac{\partial x}{\partial t} - \frac{\partial z}{\partial t} \cdot \frac{\partial x}{\partial s},$$

$$z_N = \begin{vmatrix} \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} \end{vmatrix} = \frac{\partial x}{\partial s} \cdot \frac{\partial y}{\partial t} - \frac{\partial x}{\partial t} \cdot \frac{\partial y}{\partial s}.$$

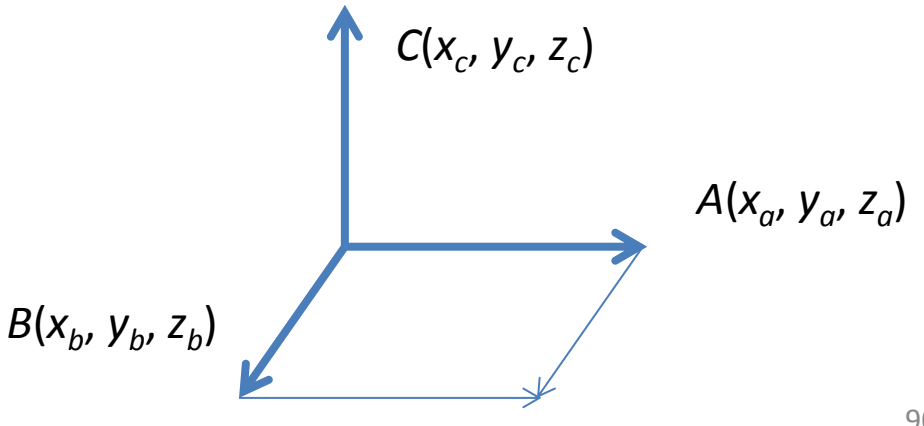
Вычисление координат вектора нормали

- В случае описания поверхности векторно-полигональной моделью для определения нормалей можно использовать методы векторной алгебры.
- **Векторное произведение векторов C=AxB -** результатом операции является вектор, перпендикулярный к плоскости параллелограмма, образованного сторонами векторов A и B, а длина вектора равна площади этого параллелограмма. В случае, когда векторы *A* и *B* являются радиус-векторами, координаты *C* вычисляются по формулам:

$$x_c = y_a \cdot z_b - z_a \cdot y_b;$$

$$y_c = z_a \cdot x_b - x_a \cdot z_b;$$

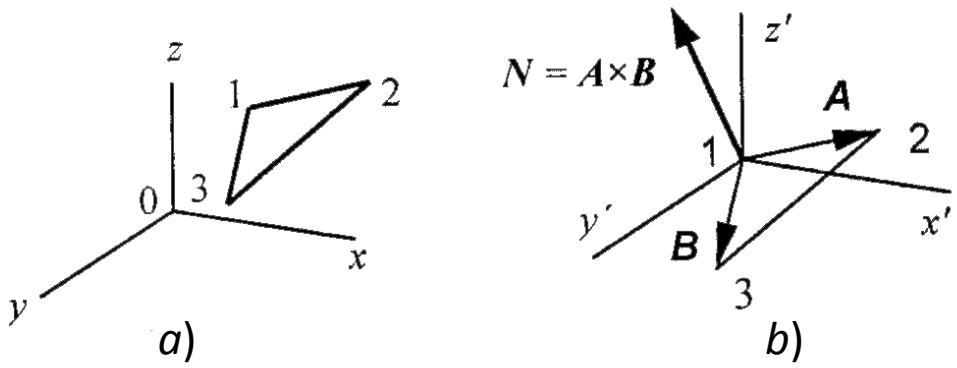
$$z_c = x_a \cdot y_b - y_a \cdot x_b;$$



Пример. Вычисление координат вектора нормали

Пусть в пространстве задана некоторая многогранная поверхность. Рассмотрим одну ее плоскую грань в виде треугольника (рис. ***a***).

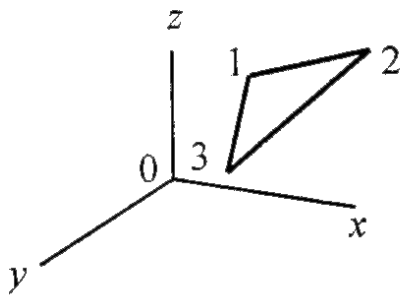
Для вычисления координат вектора нормали воспользуемся векторным произведением любых двух векторов, лежащих в плоскости грани. В качестве таких векторов могут служить и ребра грани, например, ребра 1-2 и 1-3.

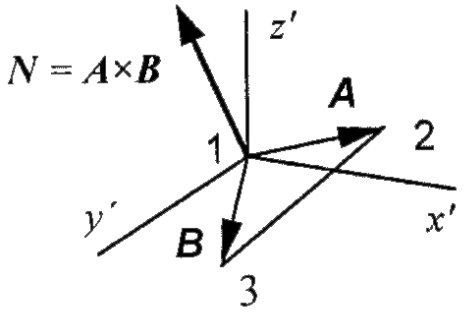


Пример. Вычисление координат вектора нормали

- Однако формулы для векторного произведения были определены только для радиус-векторов. Чтобы перейти к радиус-векторам, введем новую систему координат, центр которой совпадает с вершиной 1, а оси параллельны осям прежней системы. Координаты вершин в новой системе:

$$x_i' = x_i - x_1,$$
$$y_i' = y_i - y_1,$$
$$z_i' = z_i - z_1.$$



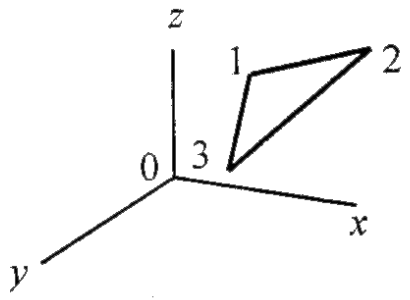


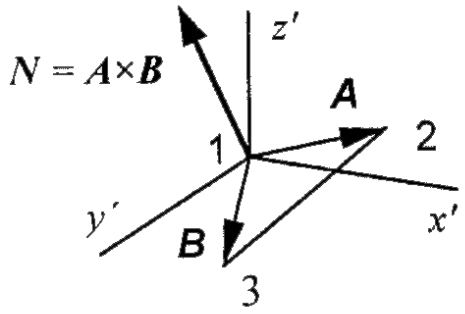
Пример. Вычисление координат вектора нормали

Теперь назовем ребро (1-2) вектором ***A,*** а ребро (1-3) – вектором ***B,*** как показано на рис. 1 ***b***. Таким образом, положение нормали к грани в пространстве будет описываться радиус-вектором ***N.*** Его координаты в системе *(x ', y', z')* выразим формулами для векторного произведения:

$$x'_N = (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1),$$
$$y'_N = (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1),$$
$$z'_N = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1).$$

Здесь использованы координаты вершин грани до переноса.





Пример. Вычисление координат вектора нормали

- Плоская грань может изображаться в различных ракурсах. В каждой конкретной ситуации необходимо выбирать направление нормали, соответствующее *видимой стороне* грани.

  Если полигональная поверхность имеет не треугольные грани, а, например, плоские четырехугольные, то расчет нормали можно выполнять по любым трем вершинам грани.

Алгоритмы, работающие в пространстве изображения

Алгоритм плавающего горизонта (АПГ)

Алгоритм плавающего горизонта (АПГ)

- АПГ используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде

$$F(x, y, z) = 0.$$

Подобные функции возникают во многих приложениях в **математике**, **технике**, **естественных науках** и других дисциплинах.

Главная идея: сведение трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат *x*, *y* или *z*.

В частности, обычно, параллельные плоскости определяются постоянными значениями z.

Алгоритм плавающего горизонта (АПГ)

Указанные параллельные плоскости определяются постоянными значениями *z*. Функция *F(x,y,z)* = 0 сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например к последовательности *y=f(x,z)* или *x=g(y,z)*,
где *z* постоянно на каждой из заданных параллельных плоскостей.

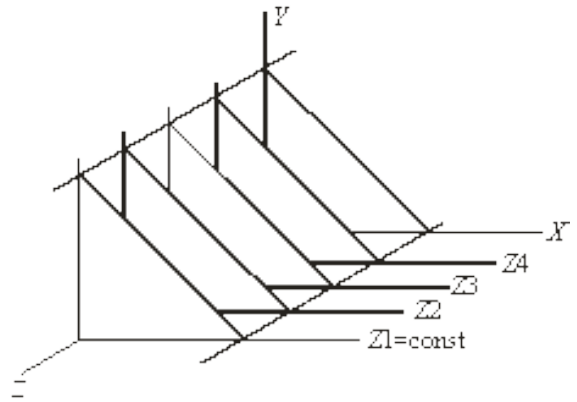
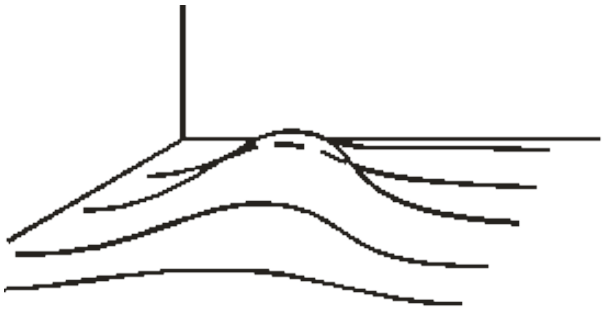


Рис. Секущие плоскости с постоянной координатой

Алгоритм плавающего горизонта (АПГ)

- Теперь поверхность складывается из последовательности кривых, лежащих в каждой из плоскостей.



- Результирующее изображения получается путем проецирования полученных кривых на плоскость $z = 0$.

Алгоритм плавающего горизонта (АПГ)

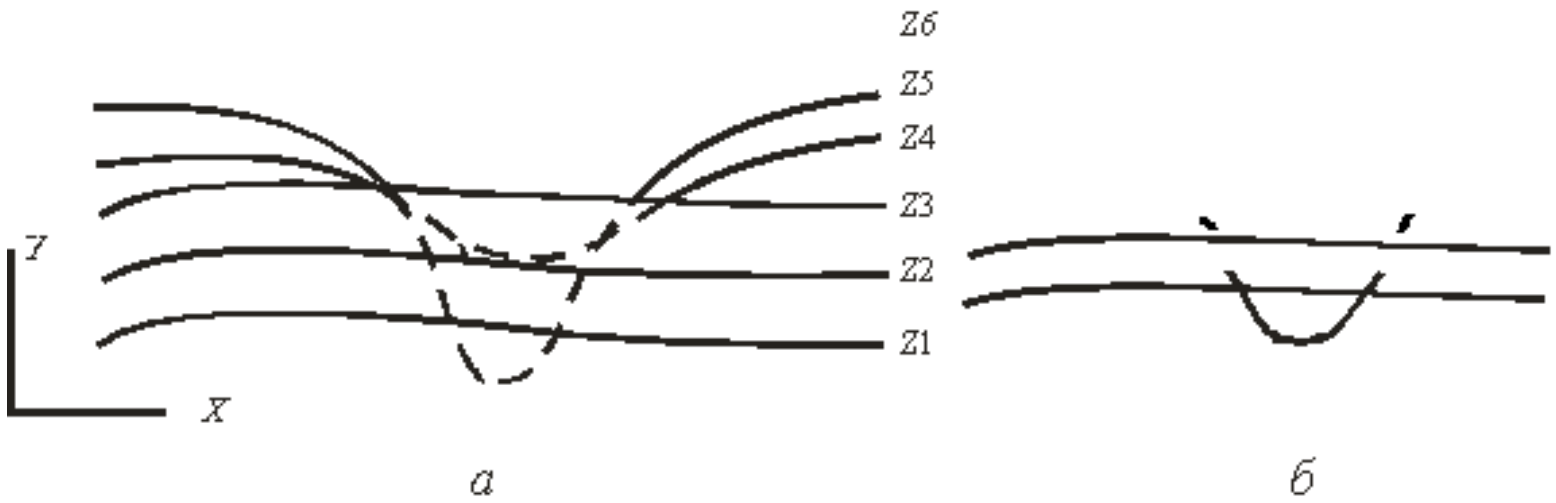
- Сначала упорядочиваются плоскости z = const по возрастанию расстояния до них от точки наблюдения.
- Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, т.е. для каждого значения координаты x в пространстве изображения определяется соответствующее значение y.
- Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше значения y для всех предыдущих кривых при этом значении x, то текущая кривая видима в этой точке; в противном случае она невидима.

Алгоритм плавающего горизонта (АПГ)

- Для хранения максимальных значений y для каждого значения x используется массив, длина которого равна числу различимых точек (пикселей) по оси x.
- Значения, хранящиеся в этом массиве, представляют собой текущие значения «горизонта». Поэтому по мере рисования каждой очередной кривой этот горизонт «всплывает».
- Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией.

Алгоритм плавающего горизонта (АПГ)

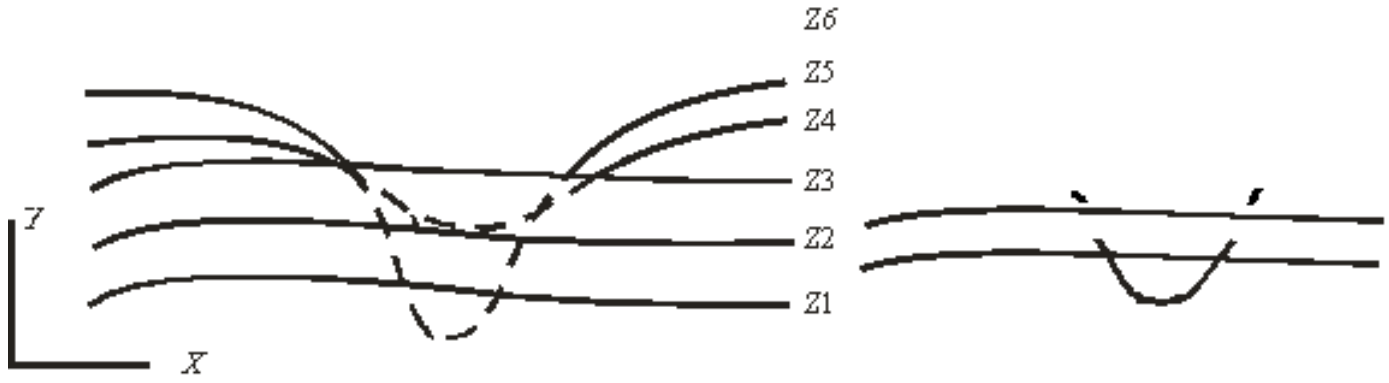
- Алгоритм работает до тех пор, пока какая-нибудь кривая не окажется ниже самой первой из кривых.



Обработка нижней стороны поверхности

Алгоритм плавающего горизонта (АПГ)

- Подобные кривые видимы и представляют собой нижнюю сторону исходной поверхности, однако алгоритм будет считать их невидимыми.
- Алгоритм становится таким: если на текущей плоскости при некотором заданном значении *x* соответствующее значение *y* на кривой больше максимума или меньше минимума по *y* для всех предыдущих кривых при этом значении *x*, то текущая кривая видима. В противном случае она невидима.
- Полученный результат показан на рис. б.

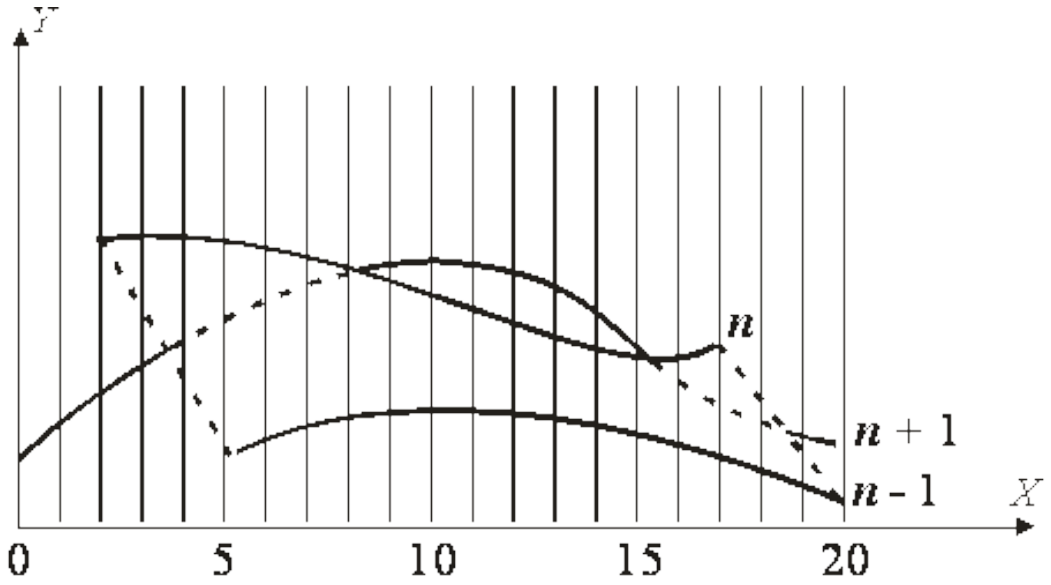


а *б*

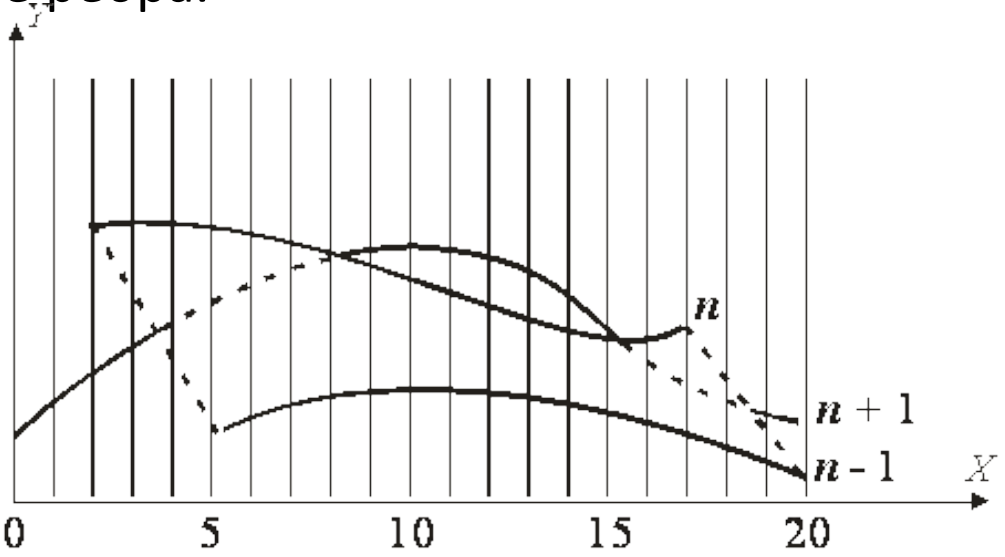
Алгоритм плавающего горизонта

Изложенный алгоритм приводит к некоторым дефектам, когда кривая, лежащая в одной из более удаленных от точки наблюдения плоскостей, появляется слева или справа из-под множества кривых, лежащих в плоскостях, которые ближе к указанной точке наблюдения.

Этот эффект продемонстрирован на рис., где уже обработанные плоскости *n*-1 и *n* расположены ближе к точке наблюдения. На рисунке показано, что получается при обработке плоскости *n*+1.



- После обработки кривых *n*-1 и n верхний горизонт для значений *x* = 0 и *x* = 1 равен начальному значению *y*; для значений *x* от 2 до 17 он равен ординатам кривой *n*; а для значений 18, 19, 20 - ординатам кривой *n*-1.
  Нижний горизонт для значений *x* = 0 и *x* = 1 равен начальному значению *y*; для значений *x* = 2, 3, 4 - ординатам кривой *n*; а для значений *x* от 5 до 20 - ординатам кривой *n*-1.
  При обработке текущей кривой (*n*+1) алгоритм объявляет ее видимой при *x* = 4. Это показано сплошной линией на рис. 5.6. Аналогичный эффект возникает и справа при *x* = 18. Такой эффект приводит к появлению зазубренных боковых ребер.
- Проблема с зазубренностью боковых ребер решается включением в массивы верхнего и нижнего горизонтов ординат, соответствующих штриховым линиям на рис. Это можно выполнить эффективно, создав ложные боковые ребра.

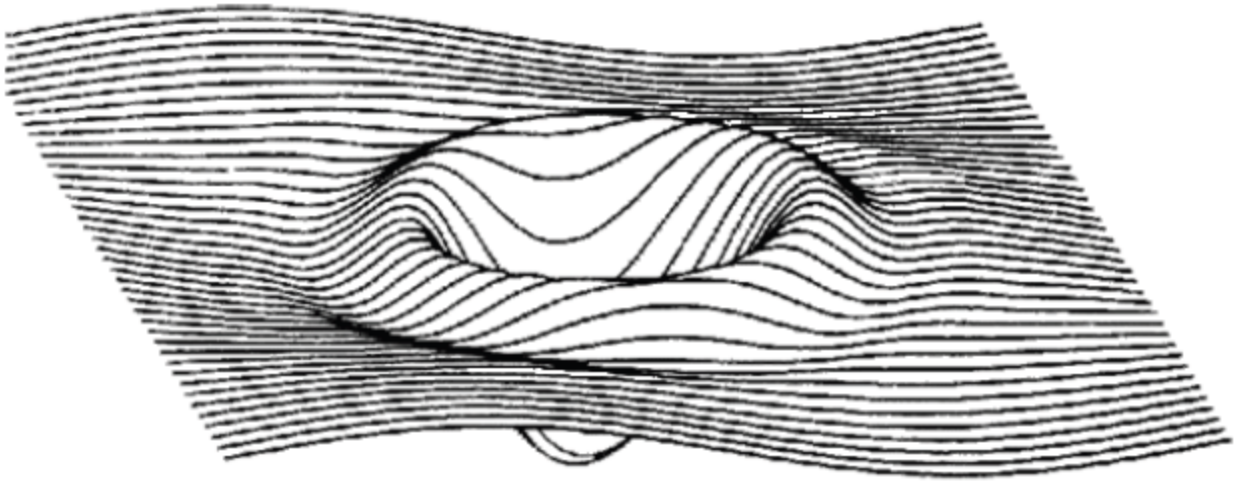


Алгоритм плавающего горизонта

глобальные массивы:
Up[W] и **Down[W]**

Функция точки(x,y):

```
if (y > Up[x])
{
  SetPixel(x,y);
  Up[x] = y;
}
if (y < Down[x])
{
  SetPixel(x,y);
  Down[x] = y;
}
```



Алгоритм, использующий z-буфер

Алгоритм, использующий z-буфер

- Был предложен **Кэтмулом**. Работает в пространстве изображения.
- Идея *z*-буфера основан на идее о буфере кадра.
  - Буфер кадра используется для запоминания интенсивности каждого пикселя в пространстве изображения,
  - z-буфер - это отдельный буфер глубины, используемый для запоминания координаты *z* или глубины каждого видимого пиксела в пространстве изображения.

- В процессе работы глубина каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в *z*-буфер.
  - Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка *z*-буфера новым значением *z*.
  - Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по *x* и *y* наибольшего значения функции *z (x, y).*

Алгоритм, использующий z-буфер

- Преимущество алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей.
- Вычислительная трудоемкости алгоритма линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в *z*-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины.
- Недостаток алгоритма - в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания.

Алгоритм, использующий z-буфер

Формальное описание алгоритма *z*-буфера таково:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить *z*-буфер минимальным значением *z*.
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке.
4. Для каждого *Пиксел*(*x,y*) в многоугольнике вычислить его глубину *z*(*x,y*).
5. Сравнить глубину *z*(*x,y*) со значением *Zбуфер*(*x,y*), хранящимся в *z*-буфере в этой же позиции.

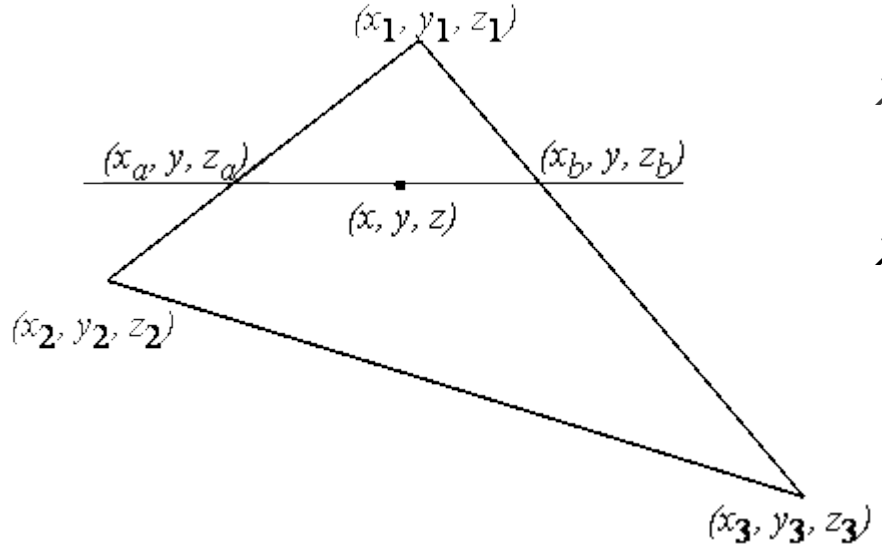
   Если *z*(*x,y*) > *Zбуфер* (*x,y*), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить *Zбуфер*(*x,y*) на *z*(*x,y*). В противном случае никаких действий не производить.

На псевдокоде алгоритм можно представить так:

```
for all objects
   for all covered pixels
      compare z
```

В качестве предварительного шага там, где это целесообразно, применяется удаление нелицевых граней.

- Если известно уравнение ребер многоугольника, то вычисление глубины каждого пиксела на сканирующей строке можно проделать пошаговым способом. Грань при этом рисуется последовательно (строка за строкой). Для нахождения необходимых значений используется линейная интерполяция.



$$x_a = x_1 + (x_2 - x_1) \cdot \frac{y - y_1}{y_2 - y_1}$$

$$x_b = x_1 + (x_3 - x_1) \cdot \frac{y - y_1}{y_3 - y_1}$$

$$z_a = z_1 + (z_2 - z_1) \cdot \frac{y - y_1}{y_2 - y_1}$$

$$z_b = z_1 + (z_3 - z_1) \cdot \frac{y - y_1}{y_3 - y_1}$$

На сканирующей строке x меняется от x_a до x_b и для каждой точки строки определяется глубина z:

$$z = z_a + (z_b - z_a) \cdot \frac{x - x_a}{x_b - x_a}$$

Алгоритм, использующий z-буфер

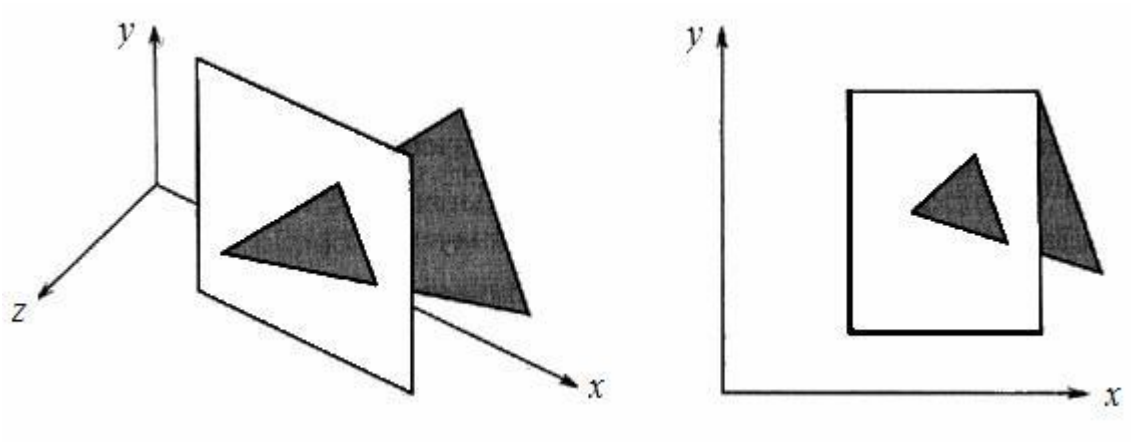
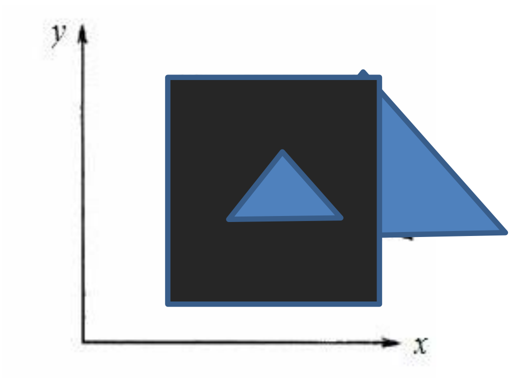


Рис. 15. Протыкающий треугольник

В начале в буфере кадра и в *z*-буфере содержатся нули. После растровой развертки прямоугольника содержимое буфера кадра будет иметь вид

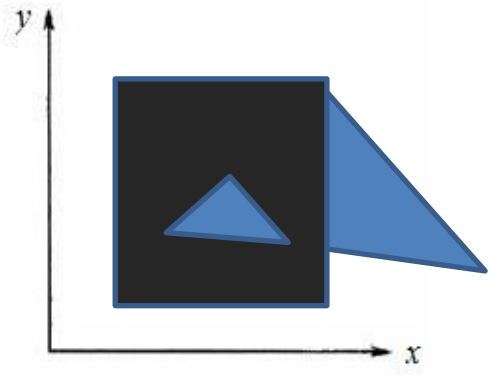


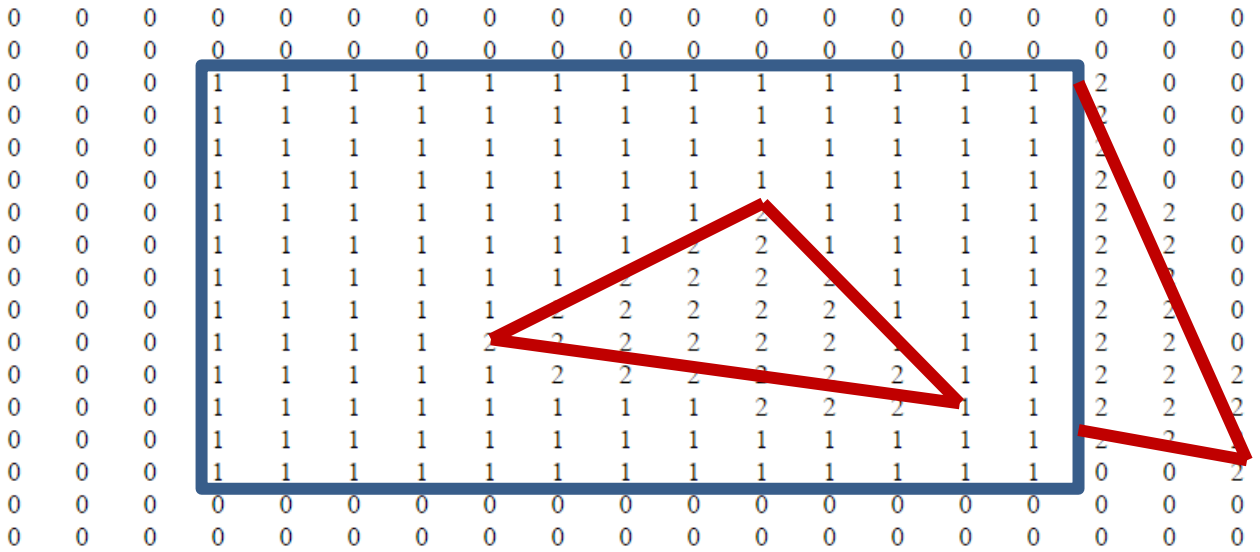
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Содержимое *z*-буфера таково:

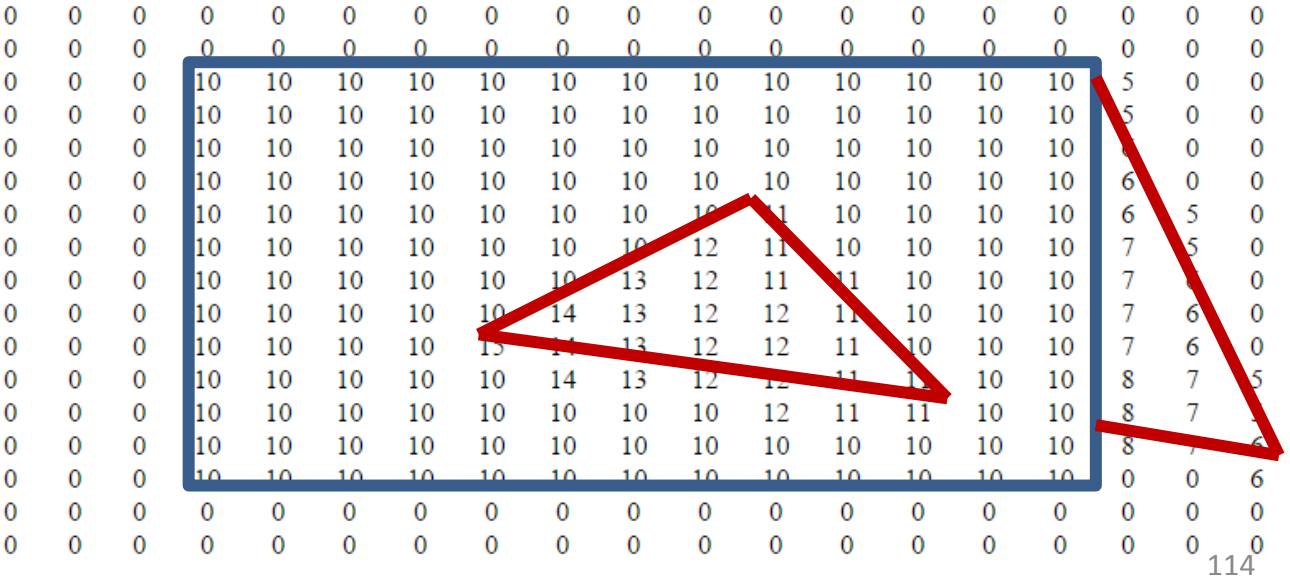
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

При обработке треугольника преобразование его в растровую форму и сравнение по глубине дает новое значение буфера кадра:





Новое содержимое *z*-буфера таково:

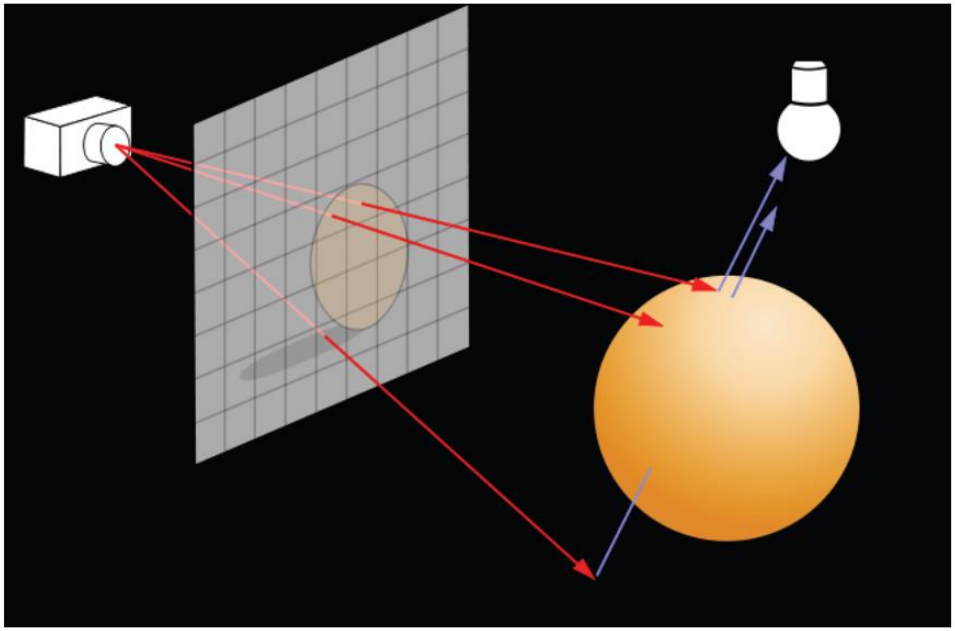


Метод трассировки лучей

Метод трассировки лучей

В этом методе для каждого пиксела картинной плоскости определяется ближайшая к нему грань, для чего через этот пиксел выпускается луч, находятся все его пересечения с гранями и среди них выбирается ближайшая. Алгоритм на псевдокоде можно кратко записать так:

```
for all pixels
   for all objects
        compare z
```



Алгоритм трассировки лучей несколько похож на алгоритм *z*-буфера, однако здесь циклы по пикселам и по объектам меняются местами.

Алгоритм Варнока

Алгоритм Варнока

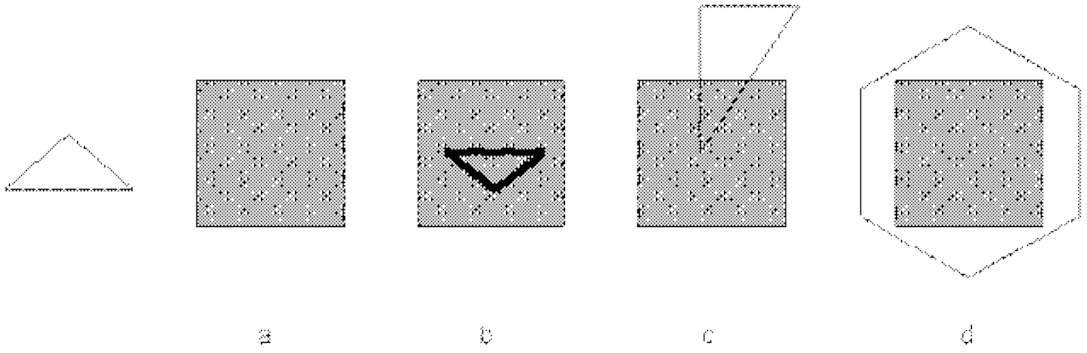
- Алгоритм Варнока основан на ***разбиении картинной плоскости на части***, для каждой из которых исходная задача может быть решена достаточно просто.
- Работает в пространстве изображения.
  - В пространстве изображения ***рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации***.
  - Если это не так, то ***окно разбивается на фрагменты*** до тех пор, пока содержимое фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения.

Алгоритм Варнока

Возможны четыре различных случая:

1. Проекция грани полностью накрывает область (рис 6.16, d);
2. Проекция грани пересекает область, но не содержится в ней полностью (рис.16, c);
3. Проекция грани целиком содержится внутри области (рис. 6.16, b);
4. Проекция грани не имеет общих внутренних точек с рассматриваемой областью (рис 16, a).

Очевидно, что в последнем случае грань вообще никак не влияет на то, что видно в данной области.



Соотношение проекции грани с подокном

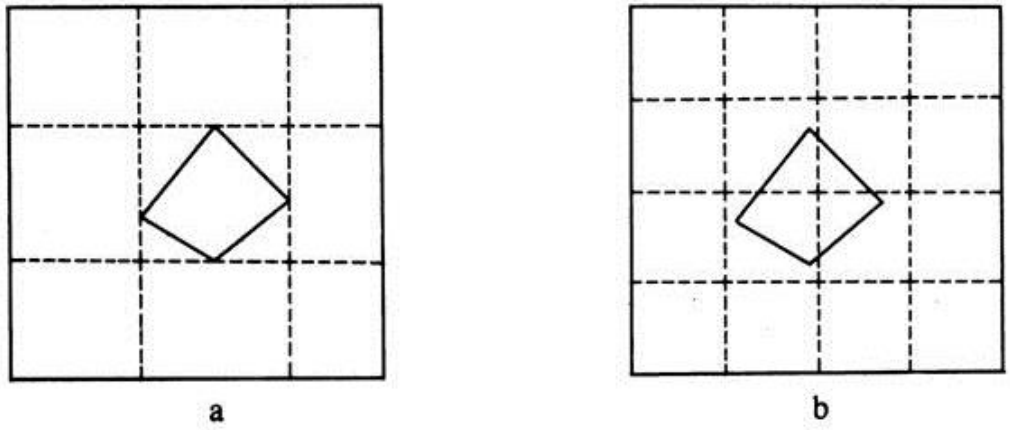
Алгоритм Варнока

- Сравнивая область с проекциями всех граней, можно выделить случаи, когда изображение, получающееся в рассматриваемой области, определяется сразу:
  - Проекция ни одной грани не попадает в область.
  - Проекция только одной грани содержится в области или пересекает область. В этом случае проекции грани разбивают всю область на две части, одна из которых соответствует этой проекции.
  - Существует грань, проекция которой полностью накрывает данную область, и эта грань расположена к картинной плоскости ближе, чем все остальные грани, проекции которых пересекают данную область. В данном случае область соответствует этой грани.

- Если ни один из рассмотренных трех случаев не имеет места, то снова разбиваем область на четыре равные части и проверяем выполнение этих условий для каждой из частей. Те части, для которых таким образом не удалось установить видимость, разбиваем снова и т. д.

Алгоритм Варнока

- Критерий останова разбиения - размер области. Как только размер области станет не больше размера одного пиксела, ближайшая грань определяется явно, как в методе трассировки лучей.
- Иногда, для устранения лестничного эффекта, процесс разбиения проводится до размеров, меньших, чем разрешение экрана, на один пиксель. При этом усредняются атрибуты подпикселей, чтобы определить атрибуты самих пикселей.
- Но! ***Негибкость*** такого способа разбиения приводят к тому, что ***количество подразбиений*** оказывается ***велико***. Можно повысить эффективность этого алгоритма, усложнив способ и критерий разбиения (рис. а), например задав ее с помощью прямоугольной объемлющей оболочки.



Способы разбиения окна

Алгоритм Вейлера-Азертона (Weiler-Atherton)

Алгоритм Вейлера-Азертона

- Разбиение картинной плоскости можно производить не только прямыми, параллельными координатным осям, но и по границам проекций граней. В результате получается точное решение задачи.
- Предлагаемый метод работает с проекциями граней на картинную плоскость:

1. Сортировка всех граней по глубине (front-to-back).
2. Затем из списка граней берется ближайшая *A*. Все остальные грани обрезаются по этой грани.

Если проекция грани *B* пересекает проекцию грани *A*, то грань *B* разбивается на части так, что каждая часть либо содержится в грани *A*, либо не имеет с ней общих внутренних точек.

Алгоритм Вейлера-Азертона

- Пролучается два множества граней:
  - F_{in} – грани, проекции которых содержатся в проекции грани A (+ сама грань A),
  - F_{out} – грани, проекции которых не имеют общих внутренних точек с проекцией грани A.
- Множество F_{in} обычно называют множеством граней, внутренних по отношению к A.
- Однако во множестве F_{in} могут быть грани, лежащие к наблюдателю ближе, чем сама грань A (при циклическом наложении граней).
- В этом случае каждая такая грань используется для разбиения всех граней из множества F_{in} (включая исходную грань A).
- Когда рекурсивное разбиение завершится, то все грани из первого множества выводятся из набора оставшихся граней (их уже ничто не может закрывать).
- Затем из набора оставшихся граней F_{out} берется очередная грань и процедура повторяется.

Алгоритм Вейлера-Азертона

Рассмотрим простейший случай для двух граней *A* и *B* (рис. 18).

- Будем считать, что грань *A* расположена ближе, чем грань *B*. Тогда на первом шаге для разбиения используется именно грань *A*. Грань *B* разбивается на две части. Часть B_1 попадает в первое множество F_{in} и, так как она лежит дальше грани *A*, удаляется.
- После этого выводится грань *A* и в списке оставшихся граней остается только грань B_2. Так как кроме нее других граней не осталось, то эта грань выводится, и на этом работа завершается.

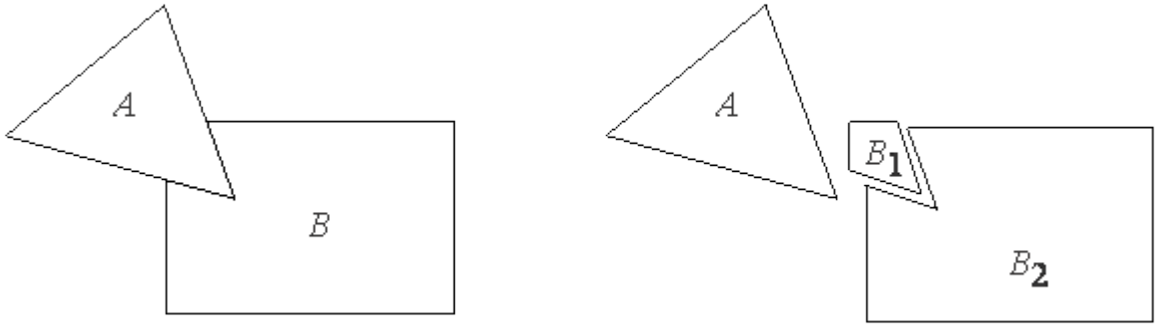


Рис. 18. Пример разбиения граней

Алгоритмы, использующие список приоритетов

Алгоритмы, использующие список приоритетов

Алгоритмы, использующие список приоритетов, пытаются получить преимущество посредством предварительной сортировки по глубине согласно расстоянию от точки наблюдения до объекта.

Если такой список окончателен, то никакие два элемента не будут взаимно перекрывать друг друга.

Эффекты прозрачности можно включить в состав алгоритма путем не полной, а частичной корректировки содержимого буфера кадра с учетом атрибутов прозрачных элементов.

Алгоритм художника

Для простых элементов сцены этот метод иногда называют алгоритмом ***художника***, поскольку он аналогичен тому способу, которым художник создает картину.
Сначала ***рисует фон***,
затем ***предметы***, лежащие на ***среднем расстоянии***,
Затем ***передний план***.
Тем самым художник решает задачу об удалении невидимых поверхностей, или задачу видимости, путем построения картины в порядке обратного приоритета.

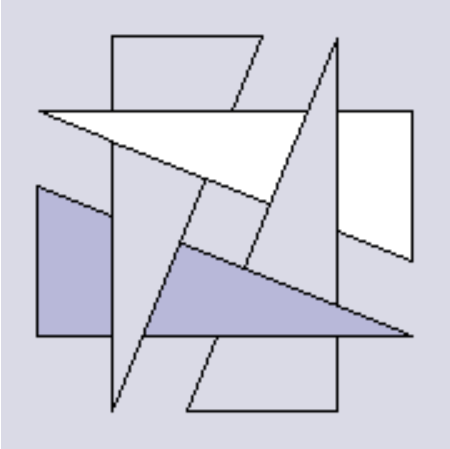
Алгоритм художника

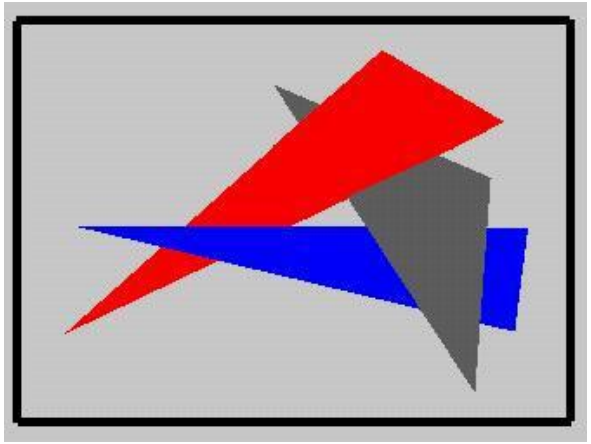
- Z-буфер обрабатывает многоугольники, поступающие в произвольном порядке, в алгоритме "художника" многоугольники сначала упорядочиваются в направлении от заднего плана к переднему.

- Когда пары многоугольников не удается достаточно просто упорядочить, они подразделяются на части до тех пор, пока получившиеся части не позволят это сделать.

Алгоритм художника. Сложности

1. При некотором расположении граней этот алгоритм вообще не может дать правильного результата - в каком порядке грани не рисуй, получится неправильно. Вот стандартные примеры.





Алгоритм художника

2. Отрисовываются все грани сцены, и при большом количестве загораживающих друг друга граней мы будем тратить большую часть времени на отрисовку невидимых в конечном итоге частей.

Алгоритм Ньюэла-Ньюэла-Санча для случая многоугольников

Алгоритм Ньюэла-Ньюэла-Санча для случая многоугольников

- Ньюэл М., Ньюэл Р. и Санч предложили специальный ***метод сортировки для разрешения конфликтов***, возникающих при создании списка приоритетов по глубине.
- В алгоритме динамически вычисляется новый список приоритетов перед обработкой каждого кадра сцены.

Алгоритм Ньюэла-Ньюэла-Санча для случая многоугольников

1. Сформировать предварительный список приоритетов по глубине, используя в качестве ключа сортировки значение z_{min} для каждого многоугольника.

   Первым в списке будет наиболее удаленный от наблюдателя многоугольник P с минимальным значением z_{min}. Следующий в списке многоугольник Q.

2. Для каждого многоугольника P из списка надо проверить его отношение с Q.

Алгоритм Ньюэла-Ньюэла-Санча для случая многоугольников

1. Если ближайшая вершина P (Pz_{max}) будет дальше от точки наблюдения, чем самая удаленная вершина Q (Qz_{min}), т. е. $Qz_{min} \geq Pz_{max}$ никакая часть P не может экранировать Q. Занести P в буфер кадра (см. рис. 16, а).

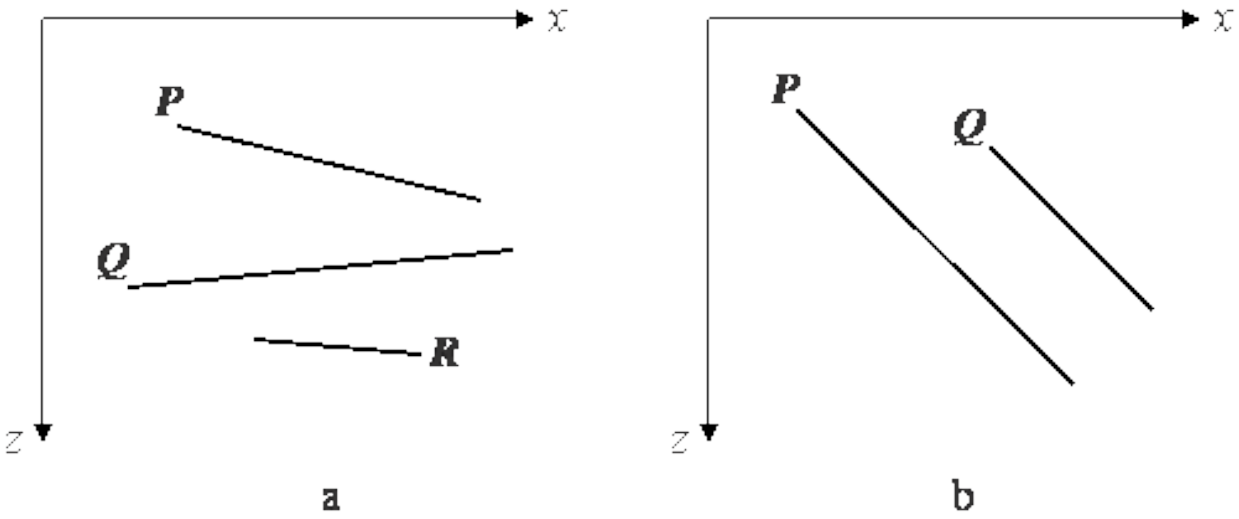


Рис. 16. Установление приоритетов для многоугольников

Алгоритм Ньюэла-Ньюэла-Санча для случая многоугольников

2. Если $Qz_{min} < Pz_{max}$, то P потенциальное экранирует не только Q, но также и любой другой многоугольник типа Q из списка, для которого $Qz_{min} < Pz_{max}$. Тем самым образуется множество $\{Q\}$.
Однако P может фактически и не экранировать ни один из этих многоугольников. Если последнее верно, то P можно заносить в буфер кадра.
Для ответа на этот вопрос используется серия тестов, следующих по возрастанию их вычислительной сложности. Эти тесты формулируются в виде вопросов.

Если ответ на один из вопросов будет положительным, то P не может экранировать $\{Q\}$. Поэтому P сразу же заносится в буфер кадра.

Алгоритм Ньюэла-Ньюэла-Санча для случая многоугольников. Тесты:

- Верно ли, что прямоугольные объемлющие оболочки *P* и *Q* не перекрываются по *x*?
- Верно ли, что прямоугольные оболочки *P* и *Q* не перекрываются по *y*?
- Верно ли, что *P* целиком лежит по ту сторону плоскости, несущей *Q*, которая расположена дальше от точки наблюдения (рис. 17, a)?
- Верно ли, что *Q* целиком лежит по ту сторону плоскости, несущей *P*, которая ближе к точке наблюдения (рис. 17, b)?
- Верно ли, что проекции *P* и *Q* не перекрываются?

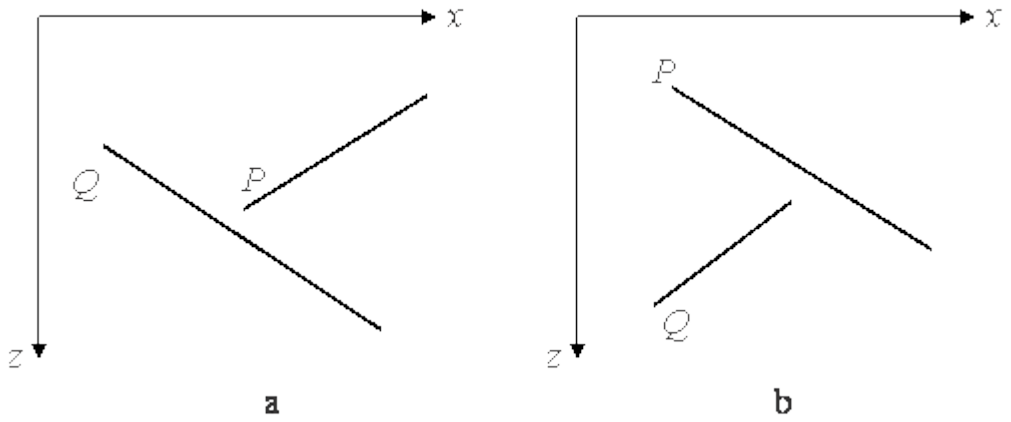


Рис. 17. Тесты для перекрывающихся многоугольников

- Каждый из этих тестов применяется к каждому элементу {*Q*}. Если ни один из них не дает положительного ответа и не заносит *P* в буфер кадра, то *P* может закрывать *Q*.
- В этом случае поменять *P* и *Q* местами, пометив позицию *Q* в списке. Повторить тесты с новым списком. Это дает положительный результат для сцены с рис. 16, b.

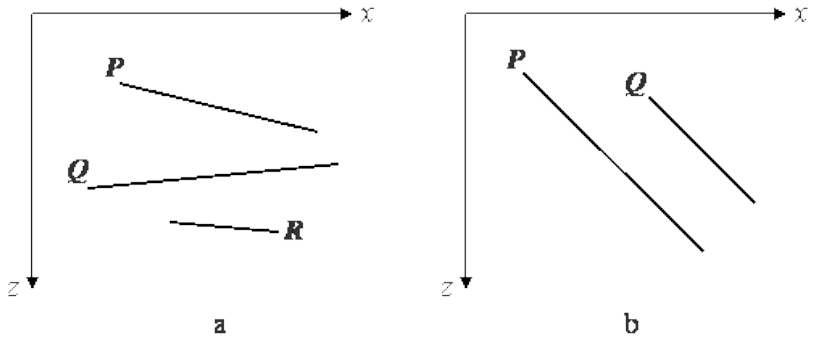


Рис. 16. Установление приоритетов для многоугольников

- Если сделана попытка вновь переставить *Q*, значит, обнаружена ситуация циклического экранирования (см. рис. 18.). В этом случае *P* разрезается плоскостью, несущей *Q*, исходный многоугольник *P* удаляется из списка, а его части заносятся в список. Затем тесты повторяются для нового списка. Этот шаг предотвращает зацикливание алгоритма.

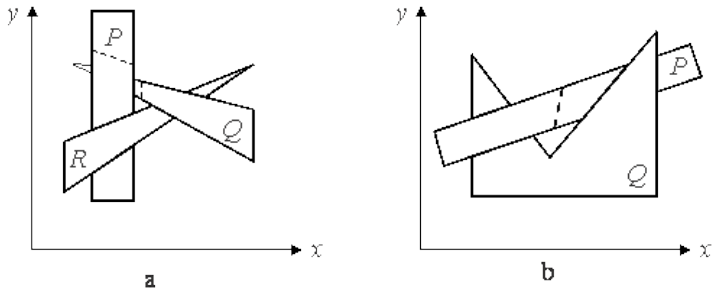


Рис. 18. Циклически перекрывающие многоугольники

Общая схема алгоритмов, использующих список приоритетов

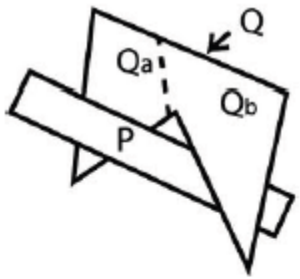
Сортировка многоугольников по оси Z (по глубине) – ближней или дальней точке

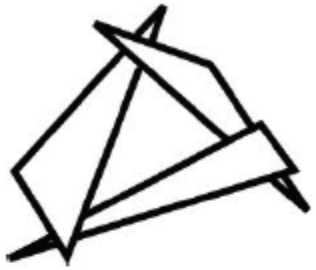
↓

Разрешение всех неоднозначностей при перекрытиях по глубине, «разрезание» при необходимости многоугольников

↓

Построение всех многоугольников от дальнего по глубине к ближнему.





текущий (дальний) мн-к P:
по каждому мн-ку Q,
с которым P пересекается по z:

Оболочки P и Q не пересекаются по x?

Оболочки P и Q не пересекаются по y?

P целиком лежит по другую сторону от Q по отношению к наблюдателю?

Q целиком лежит по одну сторону от P по отношению к наблюдателю?

Проекции P и Q на плоскость (x,y) не пересекаются?

Литература

- Д. Роджерс, Дж. Адамс. Математические основы машинной графики. М, изд-во «МИР», 2001.
- Steve Seitz, Paul Heckbert, Andy Witkin, Joel Welling, Jessica Hodgins. Implicit Surfaces and Polygons. http://www-2.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15462/web.01f/notes/
- Алексей Игнатенко. Геометрическое моделирование сплошных тел. http://graphics.cs.msu.su/ru/library/index.html